

AD A111091

LEVEL II

12

DARPA

Internet Program

DTIC
SERIALS
FEB 18 1982

E

**Internet
and
Transmission Control
Protocol Specification**

DTIC FILE COPY

This document has been approved
for public release and sale; its
distribution is unlimited.

82 02

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RFC790-796	2. GOVT ACCESSION NO. AD-A111091	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DARPA Internet Program Internet and Transmission Control Specifications		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Jonathan B. Postel		8. CONTRACT OR GRANT NUMBER(s) MDA903 81 C 0335
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90291		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE September 1981
		13. NUMBER OF PAGES 188
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) -----		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) This document is approved for public release and sale; distribution is unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer network, communication, protocol, datagram, virtual circuit, specification		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) (OVER)		

DD FORM 1473
1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)



These protocols are designed for use in an interconnected system of packet switched networks. The Internet Protocol is a basic universal protocol, and provides a uniform address space, as well as the mechanism for routing datagrams through the connected set of networks and for the fragmentation and reassembly of long datagrams if necessary for transmission through small packet networks. The Transmission Control Protocol is designed to be a highly reliable end-to-end data stream connection protocol between hosts, using mechanisms such as checksums, positive acknowledgements and timeouts with retransmission, and flow control.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

Network Working Group
Request for Comments: 790

J. Postel
ISI
September 1981

Obsoletes RFCs: 776, 770, 762, 758,
755, 750, 739, 604, 503, 433, 349
Obsoletes IENs: 127, 117, 93

ASSIGNED NUMBERS

This Network Working Group Request for Comments documents the currently assigned values from several series of numbers used in network protocol implementations. This RFC will be updated periodically, and in any case current information can be obtained from Jon Postel. The assignment of numbers is also handled by Jon. If you are developing a protocol or application that will require the use of a link, socket, port, protocol, or network number please contact Jon to receive a number assignment.

Jon Postel
USC - Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90291

phone: (213) 822-1511

ARPANET mail: POSTEL@ISIF

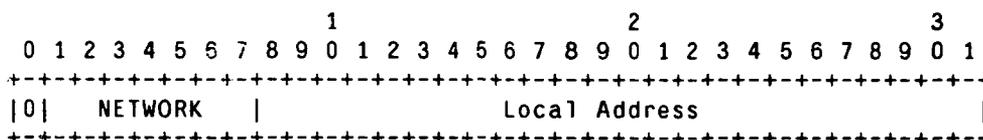
Most of the protocols mentioned here are documented in the RFC series of notes. The more prominent and more generally used are documented in the Protocol Handbook [17] prepared by the Network Information Center (NIC). Some of the items listed are undocumented. In all cases the name and mailbox of the responsible individual is indicated. In the lists that follow, a bracketed entry, e.g., [17,iii], at the right hand margin of the page indicates a reference for the listed protocol, where the number cites the document and the "iii" cites the person.

Network Numbers

ASSIGNED NETWORK NUMBERS

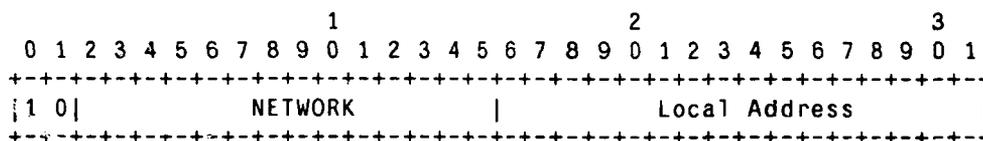
This list of network numbers is used in the internet address [33]. The Internet Protocol (IP) uses a 32 bit address and divides that address into a network part and a "rest" or local address part. The division takes 3 forms or classes.

The first type, or class a, of address has a 7-bit network number and a 24-bit local address. This allows 128 class a networks.



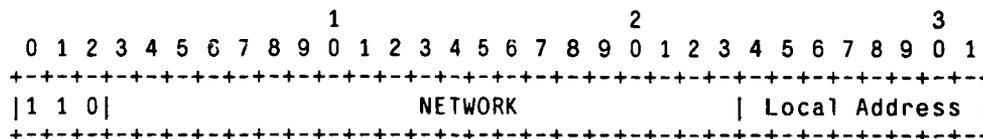
Class A Address

The second type, or class b, of address has a 14-bit network number and a 16-bit local address. This allows 16,384 class b networks.



Class B Address

The third type, or class c, of address has a 21-bit network number and a 8-bit local address. This allows 2,097,152 class c networks.



Class C Address

One notation for internet host addresses commonly used divides the 32-bit address into four 8-bit fields and specifies the value of each field as a decimal number with the fields separated by periods. For example, the internet address of ISIF is 010.020.000.002.

This notation will be used in the listing of assigned network

Network Numbers

numbers. The class a networks will have nnn.rrr.rrr.rrr, the class b networks will have nnn.nnn.rrr.rrr, and the class c networks will have nnn.nnn.nnn.rrr, where nnn represents part or all of a network number and rrr represents part or all of a local address or rest field.

Assigned Network Numbers

Class A Networks

Internet Address	Name	Network	References
-----	----	-----	-----
000.rrr.rrr.rrr		Reserved	[JBP]
001.rrr.rrr.rrr	BBN-PR	BBN Packet Radio Network	[DCA2]
002.rrr.rrr.rrr	SF-PR-1	SF Packet Radio Network (1)	[JEM]
003.rrr.rrr.rrr	BBN-RCC	BBN RCC Network	[SGC]
004.rrr.rrr.rrr	SATNET	Atlantic Satellite Network	[DM11]
005.rrr.rrr.rrr	SILL-PR	Ft. Sill Packet Radio Network	[JEM]
006.rrr.rrr.rrr	SF-PR-2	SF Packet Radio Network (2)	[JEM]
007.rrr.rrr.rrr	CHAOS	MIT CHAOS Network	[MOON]
008.rrr.rrr.rrr	CLARKNET	SATNET subnet for Clarksburg	[DM11]
009.rrr.rrr.rrr	BRAGG-PR	Ft. Bragg Packet Radio Net	[JEM]
010.rrr.rrr.rrr	ARPANET	ARPANET	[17,1,VGC]
011.rrr.rrr.rrr	UCLNET	University College London	[PK]
012.rrr.rrr.rrr	CYCLADES	CYCLADES	[VGC]
013.rrr.rrr.rrr		Unassigned	[JBP]
014.rrr.rrr.rrr	TELENET	TELENET	[VGC]
015.rrr.rrr.rrr	EPSS	British Post Office EPSS	[PK]
016.rrr.rrr.rrr	DATAPAC	DATAPAC	[VGC]
017.rrr.rrr.rrr	TRANSPAC	TRANSPAC	[VGC]
018.rrr.rrr.rrr	LCSNET	MIT LCS Network	[43,10,DDC2]
019.rrr.rrr.rrr	TYMNET	TYMNET	[VGC]
020.rrr.rrr.rrr	DC-PR	D.C. Packet Radio Network	[VGC]
021.rrr.rrr.rrr	EDN	DCEC EDN	[EC5]
022.rrr.rrr.rrr	DIALNET	DIALNET	[26,16,MRC]
023.rrr.rrr.rrr	MITRE	MITRE Cablenet	[44,APS]
024.rrr.rrr.rrr	BBN-LOCAL	BBN Local Network	[SGC]
025.rrr.rrr.rrr	RSRE-PPSN	RSRE / PPSN	[BD2]
026.rrr.rrr.rrr	AUTODIN-II	AUTODIN II	[EC5]
027.rrr.rrr.rrr	NOSC-LCCN	NOSC / LCCN	[KTP]
028.rrr.rrr.rrr	WIDEBAND	Wide Band Satellite Network	[CJW2]
029.rrr.rrr.rrr	DCN-COMSAT	COMSAT Dist. Comp. Network	[DLM1]
030.rrr.rrr.rrr	DCN-UCL	UCL Dist. Comp. Network	[PK]
031.rrr.rrr.rrr	BBN-SAT-TEST	BBN SATNET Test Network	[DM11]
032.rrr.rrr.rrr	UCL-CR1	UCL Cambridge Ring 1	[PK]
033.rrr.rrr.rrr	UCL-CR2	UCL Cambridge Ring 2	[PK]
034.rrr.rrr.rrr	MATNET	Mobile Access Terminal Net	[DM11]
035.rrr.rrr.rrr	NULL	UCL/RSRE Null Network	[BD2]

Network Numbers

036.rrr.rrr.rrr	SU-NET	Stanford University Ethernet	[MRC]
037.rrr.rrr.rrr	DECNET	Digital Equipment Network	[DRL]
038.rrr.rrr.rrr	DECNET-TEST	Test Digital Equipment Net	[DRL]
039.rrr.rrr.rrr	SRINET	SRI Local Network	[GEOF]
040.rrr.rrr.rrr	CISLNET	CISL Multics Network	[CH2]
041.rrr.rrr.rrr	BBN-LN-TEST	BBN Local Network Testbed	[KTP]
042.rrr.rrr.rrr	SINET	LLL-S1-NET	[EAK]
043.rrr.rrr.rrr	INTELPOST	COMSAT INTELPOST	[DLM1]
044.rrr.rrr.rrr	AMPRNET	Amature Radio Experiment Net	[HM]
044.rrr.rrr.rrr-126.rrr.rrr.rrr		Unassigned	[JBP]
127.rrr.rrr.rrr		Reserved	[JBP]

Class B Networks

Internet Address	Name	Network	References
-----	----	-----	-----
128.000.rrr.rrr		Reserved	[JBP]
128.001.rrr.rrr-128.254.rrr.rrr		Unassigned	[JBP]
191.255.rrr.rrr		Reserved	[JBP]

Class C Networks

Internet Address	Name	Network	References
-----	----	-----	-----
192.000.001.rrr		Reserved	[JBP]
192.000.001.rrr-223.255.254.rrr		Unassigned	[JBP]
223.255.255.rrr		Reserved	[JBP]

Other Reserved Internet Addresses

Internet Address	Name	Network	References
-----	----	-----	-----
224.000.000.000-255.255.255.255		Reserved	[JBP]

Internet Version Numbers

ASSIGNED INTERNET VERSION NUMBERS

In the Internet Protocol (IP) [33] there is a field to identify the version of the internetwork general protocol. This field is 4 bits in size.

Assigned Internet Version Numbers

Decimal	Octal	Version	References
0	0	Reserved	[JBP]
1-3	1-3	Unassigned	[JBP]
4	4	Internet Protocol	[33, JBP]
5	5	ST Datagram Mode	[20, JWF]
6-14	6-16	Unassigned	[JBP]
15	17	Reserved	[JBP]

Internet Protocol Numbers

ASSIGNED INTERNET PROTOCOL NUMBERS

In the Internet Protocol (IP) [33] there is a field, called Protocol, to identify the the next level protocol. This is an 8 bit field.

Assigned Internet Protocol Numbers

Decimal	Octal	Protocol Numbers	References
-----	-----	-----	-----
0	0	Reserved	[JBP]
1	1	ICMP	[53,JBP]
2	2	Unassigned	[JBP]
3	3	Gateway-to-Gateway	[48,49,VMS]
4	4	CMCC Gateway Monitoring Message	[18,19,DFP]
5	5	ST	[20,JWF]
6	6	TCP	[34,JBP]
7	7	UCL	[PK]
8	10	Unassigned	[JBP]
9	11	Secure	[VGC]
10	12	BBN RCC Monitoring	[VMS]
11	13	NVP	[12,DC]
12	14	PUP	[4,EAT3]
13	15	Pluribus	[RDB2]
14	16	Telenet	[RDB2]
15	17	XNET	[25,JFH2]
16	20	Chaos	[MOON]
17	21	User Datagram	[42,JBP]
18	22	Multiplexing	[13,JBP]
19	23	DCN	[DLM1]
20	24	TAC Monitoring	[55,RH6]
21-62	25-76	Unassigned	[JBP]
63	77	any local network	[JBP]
64	100	SATNET and Backroom EXPAK	[DM11]
65	101	MIT Subnet Support	[NC3]
66-68	102-104	Unassigned	[JBP]
69	105	SATNET Monitoring	[DM11]
70	106	Unassigned	[JBP]
71	107	Internet Packet Core Utility	[DM11]
72-75	110-113	Unassigned	[JBP]
76	114	Backroom SATNET Monitoring	[DM11]
77	115	Unassigned	[JBP]
78	116	WIDEBAND Monitoring	[DM11]
79	117	WIDEBAND EXPAK	[DM11]
80-254	120-376	Unassigned	[JBP]
255	377	Reserved	[JBP]

Port or Socket Numbers

ASSIGNED PORT or SOCKET NUMBERS

Ports are used in the TCP [34] and sockets are used in the AHHP [28,17] to name the ends of logical connections which carry long term conversations. For the purpose of providing services to unknown callers a service contact socket is defined. This list specifies the port or socket used by the server process as its contact socket. In the AHHP an Initial Connection Procedure ICP [39,17] is used between the user process and the server process to make the initial contact and establish the long term connections leaving the contact socket free to handle other callers. In the TCP no ICP is necessary since a port may engage in many simultaneous connections.

To the extent possible these same port assignments are used with UDP [42].

The assigned ports/sockets use a small part of the possible port/socket numbers. The assigned ports/sockets have all except the low order eight bits cleared to zero. The low order eight bits are specified here.

Socket Assignments:

General Assignments:

Decimal -----	Octal -----	Description -----
0-63	0-77	Network Wide Standard Function
64-131	100-203	Hosts Specific Functions
132-223	204-337	Reserved for Future Use
224-255	340-377	Any Experimental Function

Port or Socket Numbers

Specific Assignments:

Network Standard Functions

Decimal	Octal	Description	References
-----	-----	-----	-----
1	1	Old Telnet	[40, JBP]
3	3	Old File Transfer	[27, 11, 24, JBP]
5	5	Remote Job Entry	[6, 17, JBP]
7	7	Echo	[35, JBP]
9	11	Discard	[32, JBP]
11	13	Who is on or SYSTAT	[JBP]
13	15	Date and Time	[JBP]
15	17	Who is up or NETSTAT	[JBP]
17	21	Short Text Message	[JBP]
19	23	Character generator or TTYTST	[31, JBP]
21	25	New File Transfer	[36, JBP]
23	27	New Telnet	[41, JBP]
25	31	SMTP	[54, JBP]
27	33	NSW User System w/COMPASS FE	[14, RHT]
29	35	MSG-3 ICP	[29, RHT]
31	37	MSG-3 Authentication	[29, RHT]
33	41	Unassigned	[JBP]
35	43	IO Station Spooler	[JBP]
37	45	Time Server	[22, JBP]
39	47	Unassigned	[JBP]
41	51	Graphics	[46, 17, JBP]
42	52	Name Server	[38, JBP]
43	53	WhoIs	[JAKE]
45	55	Message Processing Module	[37, JBP]
47	57	NI FTP	[50, CJB]
49	61	RAND Network Graphics Conference	[30, MO2]
51	63	Message Generator Control	[52, DFP]
53	65	AUTODIN II FTP	[21, EC5]
55	67	ISI Graphics Language	[3, RB6]
57	71	MTP	[45, JBP]
59	73	New MIT Host Status	[SWG]
61-63	75-77	Unassigned	[JBP]

Port or Socket Numbers

Host Specific Functions

Decimal	Octal	Description	References
-----	-----	-----	-----
65	101	Unassigned	[JBP]
67	103	Datacomputer at CCA	[8,JZS]
69	105	Unassigned	[JBP]
69	105	Trivial File Transfer	[47,KRS]
71	107	NETRJS (EBCDIC) at UCLA-CCN	[5,17,RTB]
73	111	NETRJS (ASCII-68) at UCLA-CCN	[5,17,RTB]
75	113	NETRJS (ASCII-63) at UCLA-CCN	[5,17,RTB]
77	115	any private RJE server	[JBP]
79	117	Name or Finger	[23,17,KLH]
81	121	Unassigned	[JBP]
83	123	MIT ML Device	[MOON]
85	125	MIT ML Device	[MOON]
87	127	any terminal link	[JBP]
89	131	SU/MIT Telnet Gateway	[MRC]
91	133	MIT Dover Spooler	[EBM]
93	135	BBN RCC Accounting	[DT]
95	137	SUPDUP	[15,MRC]
97	141	Datacomputer Status	[8,JZS]
99	143	CADC - NIFTP via UCL	[PLH]
101	145	NPL - NIFTP via UCL	[PLH]
103	147	BNPL - NIFTP via UCL	[PLH]
105	151	CAMBRIDGE - NIFTP via UCL	[PLH]
107	153	HARWELL - NIFTP via UCL	[PLH]
109	155	SWURCC - NIFTP via UCL	[PLH]
111	157	ESSEX - NIFTP via UCL	[PLH]
113	161	RUTHERFORD - NIFTP via UCL	[PLH]
115-129	163-201	Unassigned	[JBP]
131	203	Datacomputer	[8,JZS]

Reserved for Future Use

Decimal	Octal	Description	References
-----	-----	-----	-----
132-223	204-337	Reserved	[JBP]

Port or Socket Numbers

Experimental Functions

Decimal	Octal	Description	References
-----	-----	-----	-----
224-239	340-357	Unassigned	[JBP]
241	361	NCP Measurement	[9,JBP]
243	363	Survey Measurement	[2,AV]
245	365	LINK	[7,RDB2]
247	367	TIPSRV	[RHT]
249-255	371-377	RSEXEC	[51,RHT]

ASSIGNED LINK NUMBERS

The word "link" here refers to a field in the original ARPANET Host/IMP interface leader. The link was originally defined as an 8 bit field. Some time after the ARPANET Host-to-Host (AHHP) protocol was defined and, by now, some time ago the definition of this field was changed to "Message-ID" and the length to 12 bits. The name link now refers to the high order 8 bits of this 12 bit message-id field. The low order 4 bits of the message-id field are to be zero unless specifically specified otherwise for the particular protocol used on that link. The Host/IMP interface is defined in BBN report 1822 [1].

Link Assignments:

Decimal	Octal	Description	References
-----	-----	-----	-----
0	0	AHHP Control Messages	[28,17,JBP]
1	1	Reserved	[JBP]
2-71	2-107	AHHP Regular Messages	[28,17,JBP]
72-150	110-226	Reserved	[JBP]
151	227	CHAOS Protocol	[MOON]
152	230	PARC Universal Protocol	[4,EAT3]
153	231	TIP Status Reporting	[JGH]
154	232	TIP Accounting	[JGH]
155	233	Internet Protocol (regular)	[33,JBP]
156-158	234-236	Internet Protocol (experimental)	[33,JBP]
159-191	237-277	Measurements	[9,VGC]
192-195	300-303	Unassigned	[JBP]
196-255	304-377	Experimental Protocols	[JBP]
224-255	340-377	NVP	[12,17,DC]
248-255	370-377	Network Maintenance	[JGH]

DOCUMENTS

- [1] BBN, "Specifications for the Interconnection of a Host and an IMP", Report 1822, Bolt Beranek and Newman, Cambridge, Massachusetts, May 1978.
- [2] Bhushan, A., "A Report on the Survey Project", RFC 530, NIC 17375, 22 June 1973.
- [3] Bisbey, R., D. Hollingworth, and B. Britt, "Graphics Language (version 2.1)", ISI/TM-80-18, USC/Information Sciences Institute, July 1980.
- [4] Boggs, D., J. Shoch, E. Taft, and R. Metcalfe, "PUP: An Internetwork Architecture", XEROX Palo Alto Research Center, CSL-79-10, July 1979; also in IEEE Transactions on Communication, Volume COM-28, Number 4, April 1980.
- [5] Braden, R., "NETRJS Protocol", RFC 740, NIC 42423, 22 November 1977. Also in [17].
- [6] Bressler, B., "Remote Job Entry Protocol", RFC 407, NIC 12112, 16 October 72. Also in [17].
- [7] Bressler, R., "Inter-Entity Communication -- An Experiment", RFC 441, NIC 13773, 19 January 1973.
- [8] CCA, "Datacomputer Version 5/4 User Manual", Computer Corporation of America, August 1979.
- [9] Cerf, V., "NCP Statistics", RFC 388, NIC 11360, 23 August 1972.
- [10] Clark, D., "Revision of DSP Specification", Local Network Note 9, Laboratory for Computer Science, MIT, 17 June 1977.
- [11] Clements, R., "FTPSRV -- Extensions for Tenex Paged Files", RFC 683, NIC 32251, 3 April 1975. Also in [17].
- [12] Cohen, D., "Specifications for the Network Voice Protocol (NVP)", NSC Note 68, 29 January 1976. Also as USC/Information Sciences Institute RR-75-39, March 1976, and as RFC 741, NIC 42444, 22 November 1977. Also in [17].
- [13] Cohen, D. and J. Postel, "Multiplexing Protocol", IEN 90, USC/Information Sciences Institute, May 1979.

Documents

- [14] COMPASS, "Semi-Annual Technical Report", CADD-7603-0411, Massachusetts Computer Associates, 4 March 1976. Also as, "National Software Works, Status Report No. 1", RADC-TR-76-276, Volume 1, September 1976. And COMPASS. "Second Semi-Annual Report", CADD-7608-1611, Massachusetts Computer Associates, 16 August 1976.
- [15] Crispin, M., "SUPDUP Protocol", RFC 734, NIC 41953, 7 October 1977. Also in [17].
- [16] Crispin, M. and I. Zabala, "DIALNET Protocols", Stanford University Artificial Intelligence Laboratory, July 1978.
- [17] Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.
- [18] Flood Page, D., "Gateway Monitoring Protocol", IEN 131, February 1980.
- [19] Flood Page, D., "CMCC Performance Measurement Message Formats", IEN 157, September 1980.
- [20] Forgie, J., "ST - A Proposed Internet Stream Protocol", IEN 119, M.I.T. Lincoln Laboratory, September 1979.
- [21] Forsdick, H., and A. McKenzie, "FTP Functional Specification", Bolt Beranek and Newman, Report 4051, August 1979.
- [22] Harrenstien, K., J. Postel, "Time Server", IEN 142, April 1980. Also in [17].
- [23] Harrenstien, K., "Name/Finger", RFC 742, NIC 42758, 30 December 1977. Also in [17].
- [24] Harvey, B., "One More Try on the FTP", RFC 691, NIC 32700, 6 June 1975.
- [25] Haverty, J., "XNET Formats for Internet Protocol Version 4", IEN 158, October 1980.
- [26] McCarthy, J. and L. Earnest, "DIALNET", Stanford University Artificial Intelligence Laboratory, Undated.
- [27] McKenzie, A., "File Transfer Protocol", RFC 454, NIC 14333, 16 February 1973.

Documents

- [28] McKenzie, A., "Host/Host Protocol for the ARPA Network", NIC 8246, January 1972. Also in [17].
- [29] NSW Protocol Committee, "MSG: The Interprocess Communication Facility for the National Software Works", CADD-7612-2411, Massachusetts Computer Associates, BBN 3237, Bolt Beranek and Newman, Revised 24 December 1976.
- [30] O'Brien, M., "A Network Graphical Conferencing System", RAND Corporation, N-1250-ARPA, August 1979.
- [31] Postel, J., "Character Generator Process", RFC 429, NIC 13281, 12 December 1972.
- [32] Postel, J., "Discard Process", RFC 348, NIC 10427, 30 May 1972.
- [33] Postel, J., ed., "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981.
- [34] Postel, J., ed., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, USC/Information Sciences Institute, September 1981.
- [35] Postel, J., "Echo Process", RFC 347, NIC 10426, 30 May 1972.
- [36] Postel, J., "File Transfer Protocol", RFC 765, IEN 149, June 1980.
- [37] Postel, J., "Internet Message Protocol", RFC 759, IEN 113, USC/Information Sciences Institute, August 1980.
- [38] Postel, J., "Name Server", IEN 116, USC/Information Sciences Institute, August 1979.
- [39] Postel, J., "Official Initial Connection Protocol", NIC 7101, 11 June 1971. Also in [17].
- [40] Postel, J., "Telnet Protocol", RFC 318, NIC 9348, 3 April 1972.
- [41] Postel, J., "Telnet Protocol Specification", RFC 764, IEN 148, June 1980.
- [42] Postel, J., "User Datagram Protocol", RFC 768 USC/Information Sciences Institute, August 1980.

Documents

- [43] Reed, D., "Protocols for the LCS Network", Local Network Note 3, Laboratory for Computer Science, MIT, 29 November 1976.
- [44] Skelton, A., S. Holmgren, and D. Wood, "The MITRE Cablenet Project", IEN 96, April 1979.
- [45] Sluizer, S., and J. Postel, "Mail Transfer Protocol", RFC 780, USC/Information Sciences Institute, May 1981.
- [46] Sproull, R., and E. Thomas, "A Networks Graphics Protocol", NIC 24308, 16 August 1974. Also in [17].
- [47] Sollins, K., "The TFTP Protocol (revision 2)", RFC 783, MIT/LCS, June 1981.
- [48] Strazisar, V., "Gateway Routing: An Implementation Specification", IEN 30, Bolt Berenak and Newman, April 1979.
- [49] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Berenak and Newman, August 1979.
- [50] The High Level Protocol Group, "A Network Independent File Transfer Protocol", INWG Protocol Note 86, December 1977.
- [51] Thomas, R., "A Resource Sharing Executive for the ARPANET", AFIPS Conference Proceedings, 42:155-163, NCC, 1973.
- [52] Flood Page, D., "A Simple Message Generator", IEN 172, Bolt Berenak and Newman, March 1981.
- [53] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification", RFC 792, USC/Information Sciences Institute, September 1981.
- [54] Postel, J., "Simple Mail Transfer Protocol", RFC 788, USC/Information Sciences Institute, September 1981.
- [55] Littauer, B., "A Host Monitoring Protocol", IEN 197, Bolt Berenak and Newman, September 1981.

People

PEOPLE

[DCA2]	Don Allen	BBN	Allen@BBND
[CJB]	Chris Bennett	UCL	UKSAT@ISIE
[RB6]	Richard Bisbey	ISI	Bisbey@ISIB
[RTB]	Bob Braden	UCLA	Braden@ISIA
[RDB2]	Robert Bressler	BBN	Bressler@BBNE
[EC5]	Ed Cain	DCEC	cain@EDN-Unix
[VGC]	Vint Cerf	ARPA	Cerf@ISIA
[NC3]	J. Noel Chiappa	MIT	JNC@MIT-XX
[SGC]	Steve Chipman	BBN	Chipman@BBNA
[DDC2]	David Clark	MIT	Clark@MIT-Multics
[DC]	Danny Cohen	ISI	Cohen@ISIB
[MRC]	Mark Crispin	Stanford	Admin.MRC@SU-SCORE
[BD2]	Brian Davies	RSRE	T45@ISIE
[JAKE]	Jake Feinler	SRI	Feinler@SRI-KL
[DFP]	David Flood Page	BBN	DFloodPage@BBNE
[JWF]	Jim Forgie	LL	Forgie@BBNC
[SWG]	Stu Galley	MIT	SWG@MIT-DMS
[GEOF]	Geoff Goodfellow	SRI	Geoff@DARCOM-KA
[KLH]	Ken Harrenstien	MIT	KLH@MIT-AI
[JFH2]	Jack Haverty	BBN	JHaverty@BBN-Unix
[JGH]	Jim Herman	BBN	Herman@BBNE
[PLH]	Peter Higginson	UCL	UKSAT@ISIE
[RH6]	Robert Hinden	BBN	Hinden@BBNE
[CH2]	Charles Hornig	Honeywell	Hornig@MIT-Multics
[EAK]	Earl Killian	LLL	EAK@MIT-MC
[PK]	Peter Kirstein	UCL	Kirstein@ISIA
[DRL]	David Lyons	DEC	Lyons@DEC-2136
[HM]	Hank Magnuski	---	---
[JEM]	Jim Mathis	SRI	Mathis@SRI-KL
[DM11]	Dale McNeill	BBN	DMcNeill@BBNE
[DLM1]	David Mills	COMSAT	Mills@ISIE
[MOON]	David Moon	MIT	Moon@MIT-MC
[EBM]	Eliot Moss	MIT	EBM@MIT-XX
[MO2]	Michael O'Brien	RAND	OBrien@RAND-Unix
[KTP]	Ken Pogran	BBN	Pogran@BBND
[JBP]	Jon Postel	ISI	Postel@ISIF
[JZS]	Joanne Sattely	CCA	JZS@CCA
[APS]	Anita Skelton	MITRE	skelton@MITRE
[KRS]	Karen Sollins	MIT	Sollins@MIT-XX
[VMS]	Virginia Strazisar	BBN	Strazisar@BBNA
[EAT3]	Ed Taft	XEROX	Taft.PA@PARC
[DT]	Dan Tappan	BBN	Tappan@BBNG
[RHT]	Robert Thomas	BBN	Thomas@BBNA
[AV]	Al Vezza	MIT	AV@MIT-XX
[CJW2]	Cliff Weinstein	LL	cjw@LL-11

RFC: 791

INTERNET PROTOCOL
DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

September 1981

prepared for

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

by

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

TABLE OF CONTENTS

PREFACE iii

1. INTRODUCTION 1

 1.1 Motivation 1

 1.2 Scope 1

 1.3 Interfaces 1

 1.4 Operation 2

2. OVERVIEW 5

 2.1 Relation to Other Protocols 9

 2.2 Model of Operation 5

 2.3 Function Description 7

 2.4 Gateways 9

3. SPECIFICATION 11

 3.1 Internet Header Format 11

 3.2 Discussion 23

 3.3 Interfaces 31

APPENDIX A: Examples & Scenarios 34

APPENDIX B: Data Transmission Order 39

GLOSSARY 41

REFERENCES 45

September 1981

Internet Protocol

September 1981

Internet Protocol

PREFACE

This document specifies the DoD Standard Internet Protocol. This document is based on six earlier editions of the ARPA Internet Protocol Specification, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition revises aspects of addressing, error handling, option codes, and the security, precedence, compartments, and handling restriction features of the internet protocol.

Jon Postel

Editor

September 1981

RFC: 791
Replaces: RFC 760
IENs 128, 123, 111,
80, 54, 44, 41, 28, 26

INTERNET PROTOCOL
DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

1. INTRODUCTION

1.1. Motivation

The Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks. Such a system has been called a "catenet" [1]. The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks.

1.2. Scope

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols. The internet protocol can capitalize on the services of its supporting networks to provide various types and qualities of service.

1.3. Interfaces

This protocol is called on by host-to-host protocols in an internet environment. This protocol calls on local network protocols to carry the internet datagram to the next gateway or destination host.

For example, a TCP module would call on the internet module to take a TCP segment (including the TCP header and user data) as the data portion of an internet datagram. The TCP module would provide the addresses and other parameters in the internet header to the internet module as arguments of the call. The internet module would then create an internet datagram and call on the local network interface to transmit the internet datagram.

In the ARPANET case, for example, the internet module would call on a

September 1981

Internet Protocol
Introduction

local net module which would add the 1822 leader [2] to the internet datagram creating an ARPANET message to transmit to the IMP. The ARPANET address would be derived from the internet address by the local network interface and would be the address of some host in the ARPANET, that host might be a gateway to other networks.

1.4. Operation

The internet protocol implements two basic functions: addressing and fragmentation.

The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations. The selection of a path for transmission is called routing.

The internet modules use fields in the internet header to fragment and reassemble internet datagrams when necessary for transmission through "small packet" networks.

The model of operation is that an internet module resides in each host engaged in internet communication and in each gateway that interconnects networks. These modules share common rules for interpreting address fields and for fragmenting and assembling internet datagrams. In addition, these modules (especially in gateways) have procedures for making routing decisions and other functions.

The internet protocol treats each internet datagram as an independent entity unrelated to any other internet datagram. There are no connections or logical circuits (virtual or otherwise).

The internet protocol uses four key mechanisms in providing its service: Type of Service, Time to Live, Options, and Header Checksum.

The Type of Service is used to indicate the quality of the service desired. The type of service is an abstract or generalized set of parameters which characterize the service choices provided in the networks that make up the internet. This type of service indication is to be used by gateways to select the actual transmission parameters for a particular network, the network to be used for the next hop, or the next gateway when routing an internet datagram.

The Time to Live is an indication of an upper bound on the lifetime of an internet datagram. It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the time to live reaches zero before the internet datagram reaches its destination, the internet datagram is destroyed. The time to live can be thought of as a self destruct time limit.

September 1981

Internet Protocol
Introduction

The Options provide for control functions needed or useful in some situations but unnecessary for the most common communications. The options include provisions for timestamps, security, and special routing.

The Header Checksum provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error.

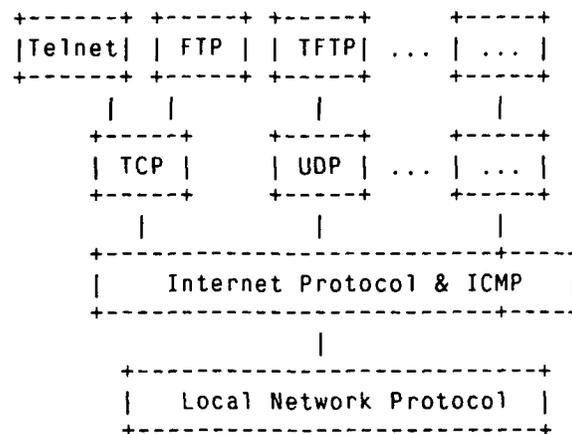
The internet protocol does not provide a reliable communication facility. There are no acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control.

Errors detected may be reported via the Internet Control Message Protocol (ICMP) [3] which is implemented in the internet protocol module.

2. OVERVIEW

2.1. Relation to Other Protocols

The following diagram illustrates the place of the internet protocol in the protocol hierarchy:



Protocol Relationships

Figure 1.

Internet protocol interfaces on one side to the higher level host-to-host protocols and on the other side to the local network protocol. In this context a "local network" may be a small network in a building or a large network such as the ARPANET.

2.2. Model of Operation

The model of operation for transmitting a datagram from one application program to another is illustrated by the following scenario:

We suppose that this transmission will involve one intermediate gateway.

The sending application program prepares its data and calls on its local internet module to send that data as a datagram and passes the destination address and other parameters as arguments of the call.

The internet module prepares a datagram header and attaches the data to it. The internet module determines a local network address for this internet address, in this case it is the address of a gateway.

Internet Protocol Overview

It sends this datagram and the local network address to the local network interface.

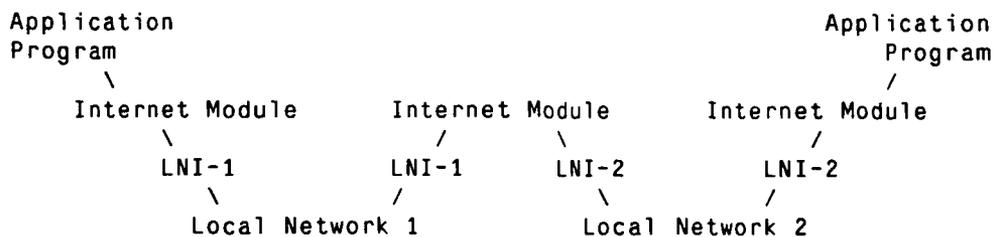
The local network interface creates a local network header, and attaches the datagram to it, then sends the result via the local network.

The datagram arrives at a gateway host wrapped in the local network header, the local network interface strips off this header, and turns the datagram over to the internet module. The internet module determines from the internet address that the datagram is to be forwarded to another host in a second network. The internet module determines a local net address for the destination host. It calls on the local network interface for that network to send the datagram.

This local network interface creates a local network header and attaches the datagram sending the result to the destination host.

At this destination host the datagram is stripped of the local net header by the local network interface and handed to the internet module.

The internet module determines that the datagram is for an application program in this host. It passes the data to the application program in response to a system call, passing the source address and other parameters as results of the call.



Transmission Path

Figure 2

2.3. Function Description

The function or purpose of Internet Protocol is to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one internet module to another until the destination is reached. The internet modules reside in hosts and gateways in the internet system. The datagrams are routed from one internet module to another through individual networks based on the interpretation of an internet address. Thus, one important mechanism of the internet protocol is the internet address.

In the routing of messages from one internet module to another, datagrams may need to traverse a network whose maximum packet size is smaller than the size of the datagram. To overcome this difficulty, a fragmentation mechanism is provided in the internet protocol.

Addressing

A distinction is made between names, addresses, and routes [4]. A name indicates what we seek. An address indicates where it is. A route indicates how to get there. The internet protocol deals primarily with addresses. It is the task of higher level (i.e., host-to-host or application) protocols to make the mapping from names to addresses. The internet module maps internet addresses to local net addresses. It is the task of lower level (i.e., local net or gateways) procedures to make the mapping from local net addresses to routes.

Addresses are fixed length of four octets (32 bits). An address begins with a network number, followed by local address (called the "rest" field). There are three formats or classes of internet addresses: in class a, the high order bit is zero, the next 7 bits are the network, and the last 24 bits are the local address; in class b, the high order two bits are one-zero, the next 14 bits are the network and the last 16 bits are the local address; in class c, the high order three bits are one-one-zero, the next 21 bits are the network and the last 8 bits are the local address.

Care must be taken in mapping internet addresses to local net addresses; a single physical host must be able to act as if it were several distinct hosts to the extent of using several distinct internet addresses. Some hosts will also have several physical interfaces (multi-homing).

That is, provision must be made for a host to have several physical interfaces to the network with each having several logical internet addresses.

Internet Protocol
Overview

Examples of address mappings may be found in "Address Mappings" [5].

Fragmentation

Fragmentation of an internet datagram is necessary when it originates in a local net that allows a large packet size and must traverse a local net that limits packets to a smaller size to reach its destination.

An internet datagram can be marked "don't fragment." Any internet datagram so marked is not to be internet fragmented under any circumstances. If internet datagram marked don't fragment cannot be delivered to its destination without fragmenting it, it is to be discarded instead.

Fragmentation, transmission and reassembly across a local network which is invisible to the internet protocol module is called intranet fragmentation and may be used [6].

The internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the identification field to ensure that fragments of different datagrams are not mixed. The fragment offset field tells the receiver the position of a fragment in the original datagram. The fragment offset and length determine the portion of the original datagram covered by this fragment. The more-fragments flag indicates (by being reset) the last fragment. These fields provide sufficient information to reassemble datagrams.

The identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.

To fragment a long internet datagram, an internet protocol module (for example, in a gateway), creates two new internet datagrams and copies the contents of the internet header fields from the long datagram into both new internet headers. The data of the long datagram is divided into two portions on a 8 octet (64 bit) boundary (the second portion might not be an integral multiple of 8 octets, but the first must be). Call the number of 8 octet blocks in the first portion NFB (for Number of Fragment Blocks). The first portion of the data is placed in the first new internet datagram, and the total length field is set to the length of the first

datagram. The more-fragments flag is set to one. The second portion of the data is placed in the second new internet datagram, and the total length field is set to the length of the second datagram. The more-fragments flag carries the same value as the long datagram. The fragment offset field of the second new internet datagram is set to the value of that field in the long datagram plus NFB.

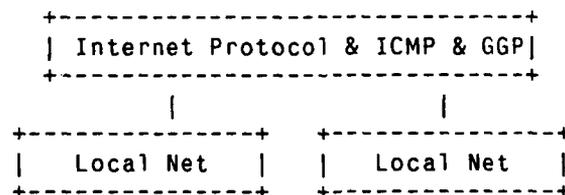
This procedure can be generalized for an n-way split, rather than the two-way split described.

To assemble the fragments of an internet datagram, an internet protocol module (for example at a destination host) combines internet datagrams that all have the same value for the four fields: identification, source, destination, and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's internet header. The first fragment will have the fragment offset zero, and the last fragment will have the more-fragments flag reset to zero.

2.4. Gateways

Gateways implement internet protocol to forward datagrams between networks. Gateways also implement the Gateway to Gateway Protocol (GGP) [7] to coordinate routing and other internet control information.

In a gateway the higher level protocols need not be implemented and the GGP functions are added to the IP module.



Gateway Protocols

Figure 3.

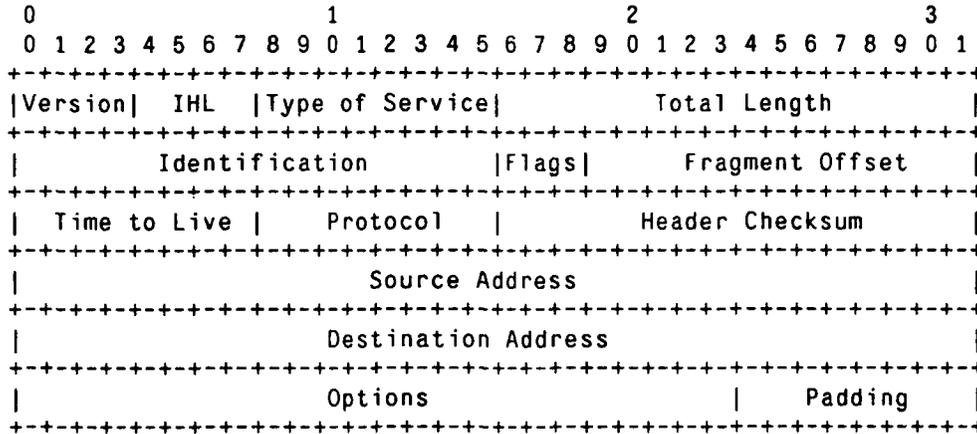
September 1981

Internet Protocol

3. SPECIFICATION

3.1. Internet Header Format

A summary of the contents of the internet header follows:



Example Internet Datagram Header

Figure 4.

Note that each tick mark represents one bit position.

Version: 4 bits

The Version field indicates the format of the internet header. This document describes version 4.

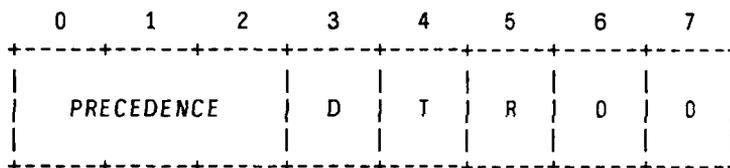
IHL: 4 bits

Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

Type of Service: 8 bits

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above a certain precedence at time of high load). The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

Bits 0-2: Precedence.
Bit 3: 0 = Normal Delay, 1 = Low Delay.
Bits 4: 0 = Normal Throughput, 1 = High Throughput.
Bits 5: 0 = Normal Reliability, 1 = High Reliability.
Bit 6-7: Reserved for Future Use.



Precedence

- 111 - Network Control
- 110 - Internetwork Control
- 101 - CRITIC/ECP
- 100 - Flash Override
- 011 - Flash
- 010 - Immediate
- 001 - Priority
- 000 - Routine

The use of the Delay, Throughput, and Reliability indications may increase the cost (in some sense) of the service. In many networks better performance for one of these parameters is coupled with worse performance on another. Except for very unusual cases at most two of these three indications should be set.

The type of service is used to specify the treatment of the datagram during its transmission through the internet system. Example mappings of the internet type of service to the actual service provided on networks such as AUTODIN II, ARPANET, SATNET, and PRNET is given in "Service Mappings" [8].

The Network Control precedence designation is intended to be used within a network only. The actual use and control of that designation is up to each network. The Internetwork Control designation is intended for use by gateway control originators only. If the actual use of these precedence designations is of concern to a particular network, it is the responsibility of that network to control the access to, and use of, those precedence designations.

Total Length: 16 bits

Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams.

The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal internet header is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of higher level protocols.

Identification: 16 bits

An identifying value assigned by the sender to aid in assembling the fragments of a datagram.

Flags: 3 bits

Various Control Flags.

Bit 0: reserved, must be zero

Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.

Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

0	1	2
+---+---+---+		
	D	M
0	F	F
+---+---+---+		

Fragment Offset: 13 bits

This field indicates where in the datagram this fragment belongs.

Internet Protocol
Specification

The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in "Assigned Numbers" [9].

Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.

The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

Source Address: 32 bits

The source address. See section 3.2.

Destination Address: 32 bits

The destination address. See section 3.2.

Options: variable

The options may appear or not in datagrams. They must be implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation.

In some environments the security option may be required in all datagrams.

The option field is variable in length. There may be zero or more options. There are two cases for the format of an option:

Case 1: A single octet of option-type.

Case 2: An option-type octet, an option-length octet, and the actual option-data octets.

The option-length octet counts the option-type octet and the option-length octet as well as the option-data octets.

The option-type octet is viewed as having 3 fields:

1 bit copied flag,
2 bits option class,
5 bits option number.

The copied flag indicates that this option is copied into all fragments on fragmentation.

0 = not copied
1 = copied

The option classes are:

0 = control
1 = reserved for future use
2 = debugging and measurement
3 = reserved for future use

The following internet options are defined:

CLASS	NUMBER	LENGTH	DESCRIPTION
0	0	-	End of Option list. This option occupies only 1 octet; it has no length octet.
0	1	-	No Operation. This option occupies only 1 octet; it has no length octet.
0	2	11	Security. Used to carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DOD requirements.
0	3	var.	Loose Source Routing. Used to route the internet datagram based on information supplied by the source.
0	9	var.	Strict Source Routing. Used to route the internet datagram based on information supplied by the source.
0	7	var.	Record Route. Used to trace the route an internet datagram takes.
0	8	4	Stream ID. Used to carry the stream identifier.
2	4	var.	Internet Timestamp.

Specific Option Definitions

End of Option List

```
+-----+
|00000000|
+-----+
Type=0
```

This option indicates the end of the option list. This might not coincide with the end of the internet header according to the internet header length. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the internet header.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

No Operation

```
+-----+
|00000001|
+-----+
  Type=1
```

This option may be used between options, for example, to align the beginning of a subsequent option on a 32 bit boundary.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

Security

This option provides a way for hosts to send security, compartmentation, handling restrictions, and TCC (closed user group) parameters. The format for this option is as follows:

```
+-----+-----+---//---+---//---+---//---+---//---+
|10000010|00001011|SSS SSS|CCC CCC|HHH HHH| TCC  |
+-----+-----+---//---+---//---+---//---+---//---+
  Type=130 Length=11
```

Security (S field): 16 bits

Specifies one of 16 levels of security (eight of which are reserved for future use).

```
00000000 00000000 - Unclassified
11110001 00110101 - Confidential
01111000 10011010 - EFTO
10111100 01001101 - MMMM
01011110 00100110 - PROG
10101111 00010011 - Restricted
11010111 10001000 - Secret
01101011 11000101 - Top Secret
00110101 11100010 - (Reserved for future use)
10011010 11110001 - (Reserved for future use)
01001101 01111000 - (Reserved for future use)
00100100 10111101 - (Reserved for future use)
00010011 01011110 - (Reserved for future use)
10001001 10101111 - (Reserved for future use)
11000100 11010110 - (Reserved for future use)
11100010 01101011 - (Reserved for future use)
```

Internet Protocol
Specification

Compartments (C field): 16 bits

An all zero value is used when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency.

Handling Restrictions (H field): 16 bits

The values for the control and release markings are alphanumeric digraphs and are defined in the Defense Intelligence Agency Manual DIAM 65-19, "Standard Security Markings".

Transmission Control Code (TCC field): 24 bits

Provides a means to segregate traffic and define controlled communities of interest among subscribers. The TCC values are trigraphs, and are available from HQ DCA Code 530.

Must be copied on fragmentation. This option appears at most once in a datagram.

Loose Source and Record Route

```

+-----+-----+-----+-----//-----+
|10000011| length | pointer|   route data   |
+-----+-----+-----+-----//-----+
Type=131

```

The loose source and record route (LSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a loose source route because the gateway or host IP is allowed to use any route of any number of other intermediate gateways to reach the next address in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Strict Source and Record Route

```

+-----+-----+-----+-----//-----+
|10001001| length | pointer|      route data      |
+-----+-----+-----+-----//-----+
Type=137

```

The strict source and record route (SSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the

Internet Protocol
Specification

recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a strict source route because the gateway or host IP must send the datagram directly to the next address in the source route through only the directly connected network indicated in the next address to reach the next gateway or host specified in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Record Route

```

+-----+-----+-----+-----//-----+
|00000111| length | pointer|   route data   |
+-----+-----+-----+-----//-----+
Type=7

```

The record route option provides a means to record the route of an internet datagram.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next area to store a route address. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A recorded route is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is

greater than the length, the recorded route data area is full. The originating host must compose this option with a large enough route data area to hold all the address expected. The size of the option does not change due to adding addresses. The initial contents of the route data area must be zero.

When an internet module routes a datagram it checks to see if the record route option is present. If it is, it inserts its own internet address as known in the environment into which this datagram is being forwarded into the recorded route beginning at the octet indicated by the pointer, and increments the pointer by four.

If the route data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the address into the recorded route. If there is some room but not enough room for a full address to be inserted, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

Not copied on fragmentation, goes in first fragment only.
Appears at most once in a datagram.

Stream Identifier

```

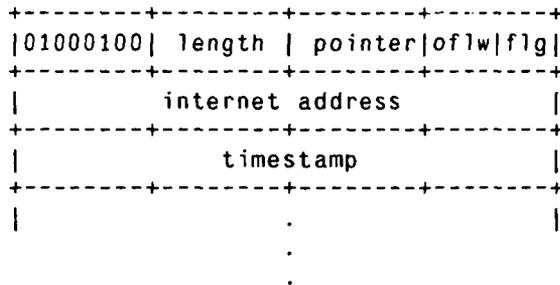
+-----+-----+-----+-----+
|10001000|00000010|   Stream ID   |
+-----+-----+-----+-----+
Type=136 Length=4

```

This option provides a way for the 16-bit SATNET stream identifier to be carried through networks that do not support the stream concept.

Must be copied on fragmentation. Appears at most once in a datagram.

Internet Timestamp



Type = 68

The Option Length is the number of octets in the option counting the type, length, pointer, and overflow/flag octets (maximum length 40).

The Pointer is the number of octets from the beginning of this option to the end of timestamps plus one (i.e., it points to the octet beginning the space for next timestamp). The smallest legal value is 5. The timestamp area is full when the pointer is greater than the length.

The Overflow (oflw) [4 bits] is the number of IP modules that cannot register timestamps due to lack of space.

The Flag (flg) [4 bits] values are

- 0 -- time stamps only, stored in consecutive 32-bit words,
- 1 -- each timestamp is preceded with internet address of the registering entity,
- 3 -- the internet address fields are prespecified. An IP module only registers its timestamp if it matches its own address with the next specified internet address.

The Timestamp is a right-justified, 32-bit timestamp in milliseconds since midnight UT. If the time is not available in milliseconds or cannot be provided with respect to midnight UT then any time may be inserted as a timestamp provided the high order bit of the timestamp field is set to one to indicate the use of a non-standard value.

The originating host must compose this option with a large enough timestamp data area to hold all the timestamp information expected. The size of the option does not change due to adding

timestamps. The initial contents of the timestamp data area must be zero or internet address/zero pairs.

If the timestamp data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the timestamp, but the overflow count is incremented by one.

If there is some room but not enough room for a full timestamp to be inserted, or the overflow count itself overflows, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

The timestamp option is not copied upon fragmentation. It is carried in the first fragment. Appears at most once in a datagram.

Padding: variable

The internet header padding is used to ensure that the internet header ends on a 32 bit boundary. The padding is zero.

3.2. Discussion

The implementation of a protocol must be robust. Each implementation must expect to interoperate with others created by different individuals. While the goal of this specification is to be explicit about the protocol there is the possibility of differing interpretations. In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear).

The basic internet service is datagram oriented and provides for the fragmentation of datagrams at gateways, with reassembly taking place at the destination internet protocol module in the destination host. Of course, fragmentation and reassembly of datagrams within a network or by private agreement between the gateways of a network is also allowed since this is transparent to the internet protocols and the higher-level protocols. This transparent type of fragmentation and reassembly is termed "network-dependent" (or intranet) fragmentation and is not discussed further here.

Internet addresses distinguish sources and destinations to the host level and provide a protocol field as well. It is assumed that each protocol will provide for whatever multiplexing is necessary within a host.

Internet Protocol Specification

Addressing

To provide for flexibility in assigning address to networks and allow for the large number of small to intermediate sized networks the interpretation of the address field is coded to specify a small number of networks with a large number of hosts, a moderate number of networks with a moderate number of hosts, and a large number of networks with a small number of hosts. In addition there is an escape code for extended addressing mode.

Address Formats:

High Order Bits	Format	Class
0	7 bits of net, 24 bits of host	a
10	14 bits of net, 16 bits of host	b
110	21 bits of net, 8 bits of host	c
111	escape to extended addressing mode	

A value of zero in the network field means this network. This is only used in certain ICMP messages. The extended addressing mode is undefined. Both of these features are reserved for future use.

The actual values assigned for network addresses is given in "Assigned Numbers" [9].

The local address, assigned by the local network, must allow for a single physical host to act as several distinct internet hosts. That is, there must be a mapping between internet host addresses and network/host interfaces that allows several internet addresses to correspond to one interface. It must also be allowed for a host to have several physical interfaces and to treat the datagrams from several of them as if they were all addressed to a single host.

Address mappings between internet addresses and addresses for ARPANET, SATNET, PRNET, and other networks are described in "Address Mappings" [5].

Fragmentation and Reassembly.

The internet identification field (ID) is used together with the source and destination address, and the protocol fields, to identify datagram fragments for reassembly.

The More Fragments flag bit (MF) is set if the datagram is not the last fragment. The Fragment Offset field identifies the fragment location, relative to the beginning of the original unfragmented datagram. Fragments are counted in units of 8 octets. The

fragmentation strategy is designed so that an unfragmented datagram has all zero fragmentation information (MF = 0, fragment offset = 0). If an internet datagram is fragmented, its data portion must be broken on 8 octet boundaries.

This format allows $2^{13} = 8192$ fragments of 8 octets each for a total of 65,536 octets. Note that this is consistent with the the datagram total length field (of course, the header is counted in the total length and not in the fragments).

When fragmentation occurs, some options are copied, but others remain with the first fragment only.

Every internet module must be able to forward a datagram of 68 octets without further fragmentation. This is because an internet header may be up to 60 octets, and the minimum fragment is 8 octets.

Every internet destination must be able to receive a datagram of 576 octets either in one piece or in fragments to be reassembled.

The fields which may be affected by fragmentation include:

- (1) options field
- (2) more fragments flag
- (3) fragment offset
- (4) internet header length field
- (5) total length field
- (6) header checksum

If the Don't Fragment flag (DF) bit is set, then internet fragmentation of this datagram is NOT permitted, although it may be discarded. This can be used to prohibit fragmentation in cases where the receiving host does not have sufficient resources to reassemble internet fragments.

One example of use of the Don't Fragment feature is to down line load a small host. A small host could have a boot strap program that accepts a datagram stores it in memory and then executes it.

The fragmentation and reassembly procedures are most easily described by examples. The following procedures are example implementations.

General notation in the following pseudo programs: " \leq " means "less than or equal", " \neq " means "not equal", " $=$ " means "equal", " \leftarrow " means "is set to". Also, "x to y" includes x and excludes y; for example, "4 to 7" would include 4, 5, and 6 (but not 7).

An Example Fragmentation Procedure

The maximum sized datagram that can be transmitted through the next network is called the maximum transmission unit (MTU).

If the total length is less than or equal the maximum transmission unit then submit this datagram to the next step in datagram processing; otherwise cut the datagram into two fragments, the first fragment being the maximum size, and the second fragment being the rest of the datagram. The first fragment is submitted to the next step in datagram processing, while the second fragment is submitted to this procedure in case it is still too large.

Notation:

FO - Fragment Offset
IHL - Internet Header Length
DF - Don't Fragment flag
MF - More Fragments flag
TL - Total Length
OFO - Old Fragment Offset
OIHL - Old Internet Header Length
OMF - Old More Fragments flag
OTL - Old Total Length
NFB - Number of Fragment Blocks
MTU - Maximum Transmission Unit

Procedure:

IF TL \leq MTU THEN Submit this datagram to the next step
in datagram processing ELSE IF DF = 1 THEN discard the
datagram ELSE

To produce the first fragment:

- (1) Copy the original internet header;
- (2) $OIHL \leftarrow IHL$; $OTL \leftarrow TL$; $OFO \leftarrow FO$; $OMF \leftarrow MF$;
- (3) $NFB \leftarrow (MTU - IHL * 4) / 8$;
- (4) Attach the first $NFB * 8$ data octets;
- (5) Correct the header:
 $MF \leftarrow 1$; $TL \leftarrow (IHL * 4) + (NFB * 8)$;
Recompute Checksum;
- (6) Submit this fragment to the next step in
datagram processing;

To produce the second fragment:

- (7) Selectively copy the internet header (some options
are not copied, see option definitions);
- (8) Append the remaining data;
- (9) Correct the header:
 $IHL \leftarrow (((OIHL * 4) - (\text{length of options not copied})) + 3) / 4$;

```
TL <- OTL - NFB*8 - (OIHl-IHL)*4);
FO <- OFO + NFB; MF <- OMF; Recompute Checksum;
(10) Submit this fragment to the fragmentation test; DONE.
```

In the above procedure each fragment (except the last) was made the maximum allowable size. An alternative might produce less than the maximum size datagrams. For example, one could implement a fragmentation procedure that repeatedly divided large datagrams in half until the resulting fragments were less than the maximum transmission unit size.

An Example Reassembly Procedure

For each datagram the buffer identifier is computed as the concatenation of the source, destination, protocol, and identification fields. If this is a whole datagram (that is both the fragment offset and the more fragments fields are zero), then any reassembly resources associated with this buffer identifier are released and the datagram is forwarded to the next step in datagram processing.

If no other fragment with this buffer identifier is on hand then reassembly resources are allocated. The reassembly resources consist of a data buffer, a header buffer, a fragment block bit table, a total data length field, and a timer. The data from the fragment is placed in the data buffer according to its fragment offset and length, and bits are set in the fragment block bit table corresponding to the fragment blocks received.

If this is the first fragment (that is the fragment offset is zero) this header is placed in the header buffer. If this is the last fragment (that is the more fragments field is zero) the total data length is computed. If this fragment completes the datagram (tested by checking the bits set in the fragment block table), then the datagram is sent to the next step in datagram processing; otherwise the timer is set to the maximum of the current timer value and the value of the time to live field from this fragment; and the reassembly routine gives up control.

If the timer runs out, the all reassembly resources for this buffer identifier are released. The initial setting of the timer is a lower bound on the reassembly waiting time. This is because the waiting time will be increased if the Time to Live in the arriving fragment is greater than the current timer value but will not be decreased if it is less. The maximum this timer value could reach is the maximum time to live (approximately 4.25 minutes). The current recommendation for the initial timer setting is 15 seconds. This may be changed as experience with

this protocol accumulates. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the transmission medium; that is, data rate times timer value equals buffer size (e.g., 10Kb/s X 15s = 150Kb).

Notation:

FO - Fragment Offset
IHL - Internet Header Length
MF - More Fragments flag
TTL - Time To Live
NFB - Number of Fragment Blocks
TL - Total Length
TDL - Total Data Length
BUFID - Buffer Identifier
RCVBT - Fragment Received Bit Table
TLB - Timer Lower Bound

Procedure:

```
(1) BUFID <- source|destination|protocol|identification;
(2) IF FO = 0 AND MF = 0
(3)   THEN IF buffer with BUFID is allocated
(4)         THEN flush all reassembly for this BUFID;
(5)         Submit datagram to next step; DONE.
(6)   ELSE IF no buffer with BUFID is allocated
(7)         THEN allocate reassembly resources
              with BUFID;
              TIMER <- TLB; TDL <- 0;
(8)         put data from fragment into data buffer with
              BUFID from octet FO*8 to
              octet (TL-(IHL*4))+FO*8;
(9)         set RCVBT bits from FO
              to FO+((TL-(IHL*4)+7)/8);
(10)        IF MF = 0 THEN TDL <- TL-(IHL*4)+(FO*8)
(11)        IF FO = 0 THEN put header in header buffer
(12)        IF TDL # 0
(13)        AND all RCVBT bits from 0
              to (TDL+7)/8 are set
(14)        THEN TL <- TDL+(IHL*4)
(15)        Submit datagram to next step;
(16)        free all reassembly resources
              for this BUFID; DONE.
(17)        TIMER <- MAX(TIMER,TTL);
(18)        give up until next fragment or timer expires;
(19) timer expires: flush all reassembly with this BUFID; DONE.
```

In the case that two or more fragments contain the same data

either identically or through a partial overlap, this procedure will use the more recently arrived copy in the data buffer and datagram delivered.

Identification

The choice of the Identifier for a datagram is based on the need to provide a way to uniquely identify the fragments of a particular datagram. The protocol module assembling fragments judges fragments to belong to the same datagram if they have the same source, destination, protocol, and Identifier. Thus, the sender must choose the Identifier to be unique for this source, destination pair and protocol for the time the datagram (or any fragment of it) could be alive in the internet.

It seems then that a sending protocol module needs to keep a table of Identifiers, one entry for each destination it has communicated with in the last maximum packet lifetime for the internet.

However, since the Identifier field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination.

It is appropriate for some higher level protocols to choose the identifier. For example, TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment.

Type of Service

The type of service (TOS) is for internet service quality selection. The type of service is specified along the abstract parameters precedence, delay, throughput, and reliability. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram traverses.

Precedence. An independent measure of the importance of this datagram.

Delay. Prompt delivery is important for datagrams with this indication.

Throughput. High data rate is important for datagrams with this indication.

Internet Protocol
Specification

Reliability. A higher level of effort to ensure delivery is important for datagrams with this indication.

For example, the ARPANET has a priority bit, and a choice between "standard" messages (type 0) and "uncontrolled" messages (type 3), (the choice between single packet and multipacket messages can also be considered a service parameter). The uncontrolled messages tend to be less reliably delivered and suffer less delay. Suppose an internet datagram is to be sent through the ARPANET. Let the internet type of service be given as:

Precedence: 5
Delay: 0
Throughput: 1
Reliability: 1

In this example, the mapping of these parameters to those available for the ARPANET would be to set the ARPANET priority bit on since the Internet precedence is in the upper half of its range, to select standard messages since the throughput and reliability requirements are indicated and delay is not. More details are given on service mappings in "Service Mappings" [8].

Time to Live

The time to live is set by the sender to the maximum time the datagram is allowed to be in the internet system. If the datagram is in the internet system longer than the time to live, then the datagram must be destroyed.

This field must be decreased at each point that the internet header is processed to reflect the time spent processing the datagram. Even if no local information is available on the time actually spent, the field must be decremented by 1. The time is measured in units of seconds (i.e. the value 1 means one second). Thus, the maximum time to live is 255 seconds or 4.25 minutes. Since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Some higher level reliable connection protocols are based on assumptions that old duplicate datagrams will not arrive after a certain time elapses. The TTL is a way for such protocols to have an assurance that their assumption is met.

Options

The options are optional in each datagram, but required in implementations. That is, the presence or absence of an option is the choice of the sender, but each internet module must be able to parse every option. There can be several options present in the option field.

The options might not end on a 32-bit boundary. The internet header must be filled out with octets of zeros. The first of these would be interpreted as the end-of-options option, and the remainder as internet header padding.

Every internet module must be able to act on every option. The Security Option is required if classified, restricted, or compartmented traffic is to be passed.

Checksum

The internet header checksum is recomputed if the internet header is changed. For example, a reduction of the time to live, additions or changes to internet options, or due to fragmentation. This checksum at the internet level is intended to protect the internet header fields from transmission errors.

There are some applications where a few data bit errors are acceptable while retransmission delays are not. If the internet protocol enforced data correctness such applications could not be supported.

Errors

Internet protocol errors may be reported via the ICMP messages [3].

3.3. Interfaces

The functional description of user interfaces to the IP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different IP implementations may have different user interfaces. However, all IPs must provide a certain minimum set of services to guarantee that all IP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all IP implementations.

Internet protocol interfaces on one side to the local network and on the other side to either a higher level protocol or an application program. In the following, the higher level protocol or application

Internet Protocol
Specification

program (or even a gateway program) will be called the "user" since it is using the internet module. Since internet protocol is a datagram protocol, there is minimal memory or state maintained between datagram transmissions, and each call on the internet protocol module by the user supplies all information necessary for the IP to perform the service requested.

An Example Upper Level Interface

The following two example calls satisfy the requirements for the user to internet protocol module communication ("=>" means returns):

SEND (src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt => result)

where:

src = source address
dst = destination address
prot = protocol
TOS = type of service
TTL = time to live
BufPTR = buffer pointer
len = length of buffer
Id = Identifier
DF = Don't Fragment
opt = option data
result = response
 OK = datagram sent ok
 Error = error in arguments or local network error

Note that the precedence is included in the TOS and the security/compartments is passed as an option.

RCV (BufPTR, prot, => result, src, dst, TOS, len, opt)

where:

BufPTR = buffer pointer
prot = protocol
result = response
 OK = datagram received ok
 Error = error in arguments
len = length of buffer
src = source address
dst = destination address
TOS = type of service
opt = option data

When the user sends a datagram, it executes the SEND call supplying all the arguments. The internet protocol module, on receiving this call, checks the arguments and prepares and sends the message. If the arguments are good and the datagram is accepted by the local network, the call returns successfully. If either the arguments are bad, or the datagram is not accepted by the local network, the call returns unsuccessfully. On unsuccessful returns, a reasonable report must be made as to the cause of the problem, but the details of such reports are up to individual implementations.

When a datagram arrives at the internet protocol module from the local network, either there is a pending RECV call from the user addressed or there is not. In the first case, the pending call is satisfied by passing the information from the datagram to the user. In the second case, the user addressed is notified of a pending datagram. If the user addressed does not exist, an ICMP error message is returned to the sender, and the data is discarded.

The notification of a user may be via a pseudo interrupt or similar mechanism, as appropriate in the particular operating system environment of the implementation.

A user's RECV call may then either be immediately satisfied by a pending datagram, or the call may be pending until a datagram arrives.

The source address is included in the send call in case the sending host has several addresses (multiple physical connections or logical addresses). The internet module must check to see that the source address is one of the legal address for this host.

An implementation may also allow or require a call to the internet module to indicate interest in or reserve exclusive use of a class of datagrams (e.g., all those with a certain value in the protocol field).

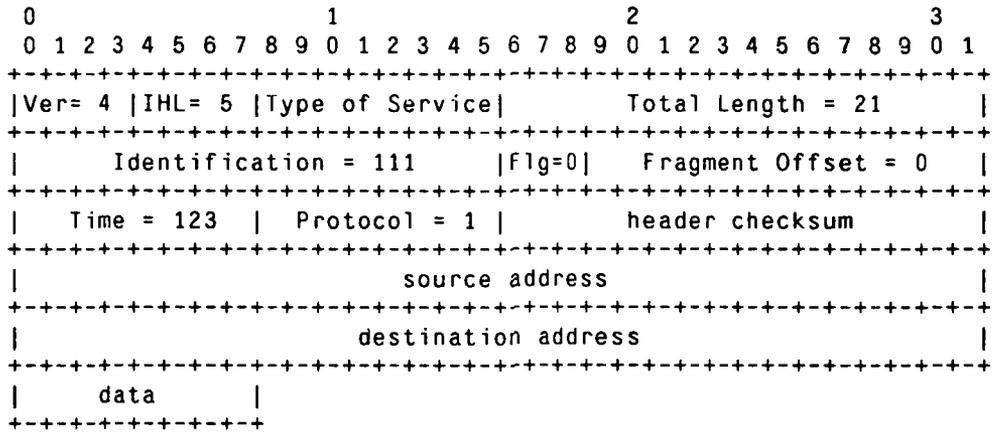
This section functionally characterizes a USER/IP interface. The notation used is similar to most procedure of function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs), or any other form of interprocess communication.

Internet Protocol

APPENDIX A: Examples & Scenarios

Example 1:

This is an example of the minimal data carrying internet datagram:



Example Internet Datagram

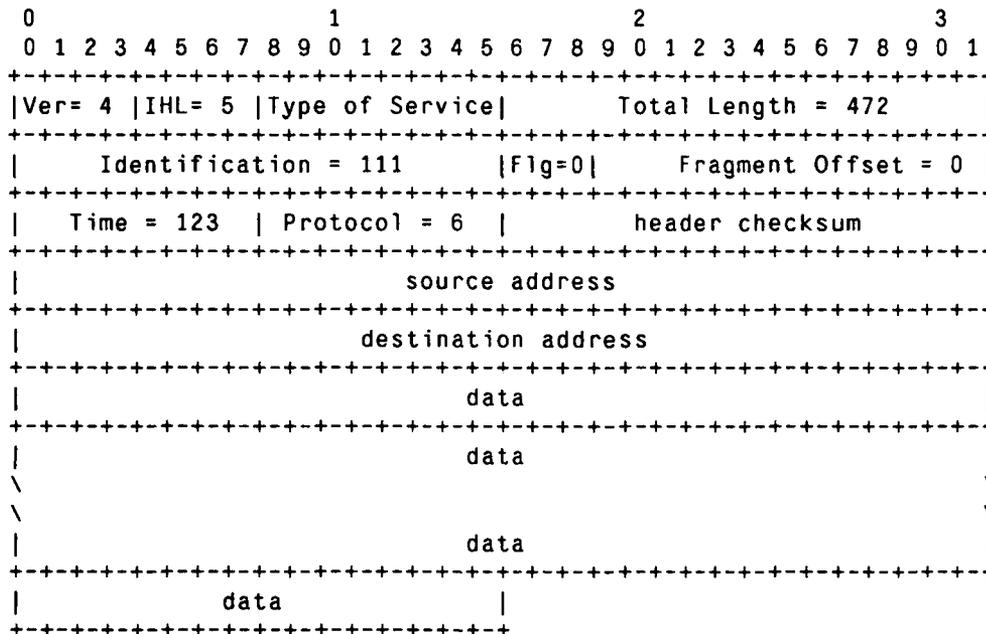
Figure 5.

Note that each tick mark represents one bit position.

This is a internet datagram in version 4 of internet protocol; the internet header consists of five 32 bit words, and the total length of the datagram is 21 octets. This datagram is a complete datagram (not a fragment).

Example 2:

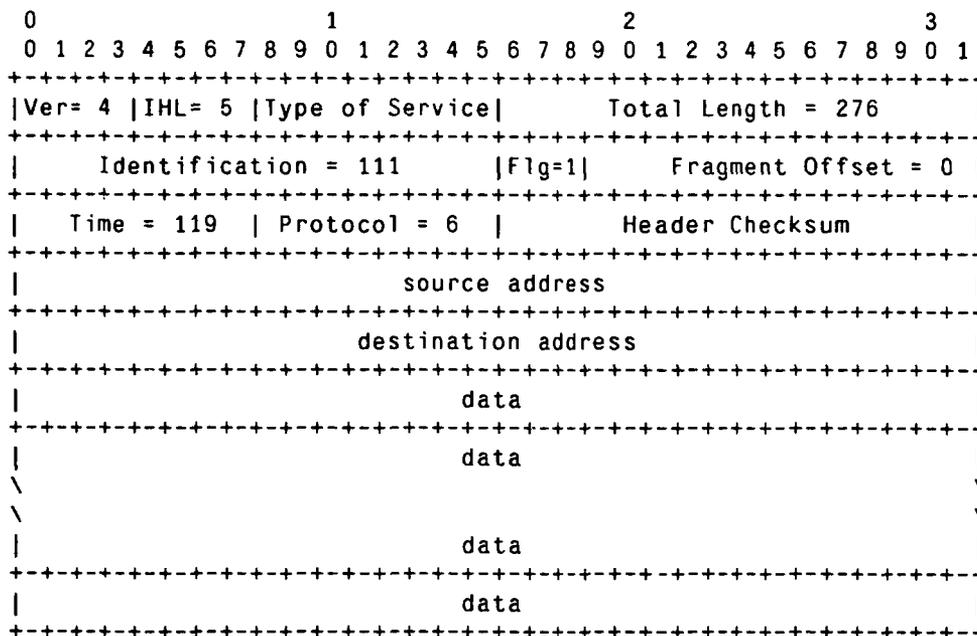
In this example, we show first a moderate size internet datagram (452 data octets), then two internet fragments that might result from the fragmentation of this datagram if the maximum sized transmission allowed were 280 octets.



Example Internet Datagram

Figure 6.

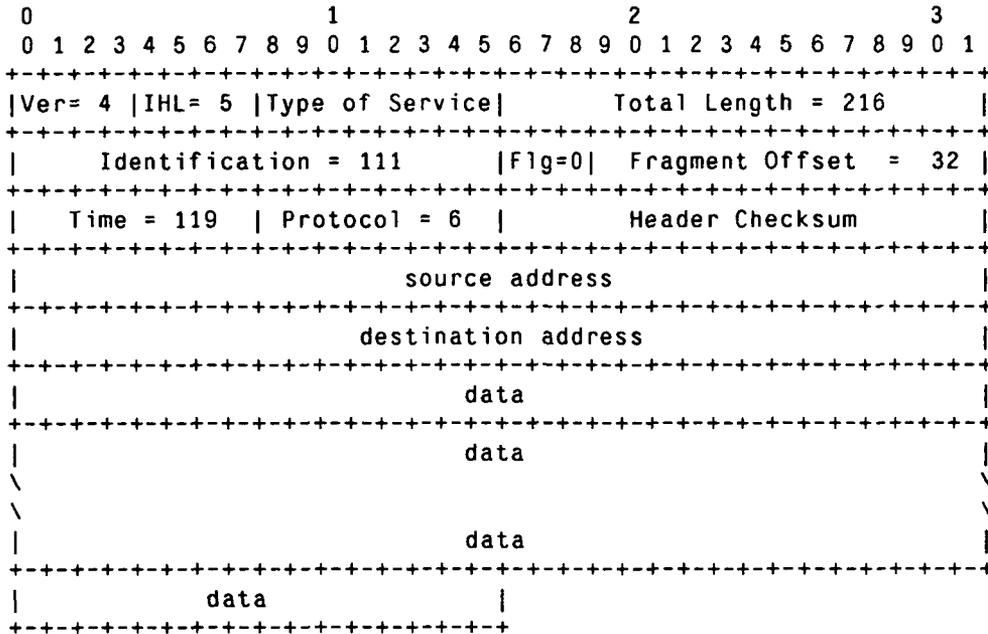
Now the first fragment that results from splitting the datagram after 256 data octets.



Example Internet Fragment

Figure 7.

And the second fragment.

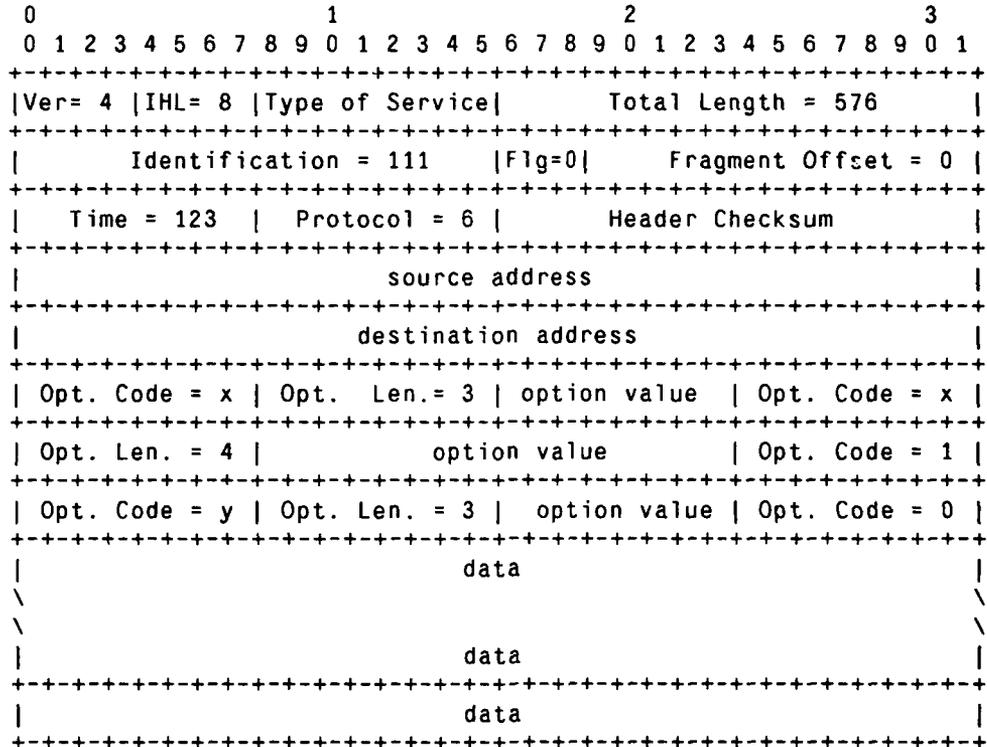


Example Internet Fragment

Figure 8.

Example 3:

Here, we show an example of a datagram containing options:



Example Internet Datagram

Figure 9.

September 1981

Internet Protocol

GLOSSARY

1822

BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ARPANET leader

The control information on an ARPANET message at the host-IMP interface.

ARPANET message

The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet

A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

Destination

The destination address, an internet header field.

DF

The Don't Fragment bit carried in the flags field.

Flags

An internet header field carrying various control flags.

Fragment Offset

This internet header field indicates where in the internet datagram a fragment belongs.

GGP

Gateway to Gateway Protocol, the protocol used primarily between gateways to control routing and other gateway functions.

header

Control information at the beginning of a message, segment, datagram, packet or block of data.

ICMP

Internet Control Message Protocol, implemented in the internet module, the ICMP is used from gateways to hosts and between hosts to report errors and make routing suggestions.

Internet Protocol
Glossary

Identification

An internet header field carrying the identifying value assigned by the sender to aid in assembling the fragments of a datagram.

IHL

The internet header field Internet Header Length is the length of the internet header measured in 32 bit words.

IMP

The Interface Message Processor, the packet switch of the ARPANET.

Internet Address

A four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.

internet datagram

The unit of data exchanged between a pair of internet modules (includes the internet header).

internet fragment

A portion of the data of an internet datagram with an internet header.

Local Address

The address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.

MF

The More-Fragments Flag carried in the internet header flags field.

module

An implementation, usually in software, of a protocol or other procedure.

more-fragments flag

A flag indicating whether or not this internet datagram contains the end of an internet datagram, carried in the internet header Flags field.

NFB

The Number of Fragment Blocks in a the data portion of an internet fragment. That is, the length of a portion of data measured in 8 octet units.

- octet An eight bit byte.
- Options The internet header Options field may contain several options, and each option may be several octets in length.
- Padding The internet header Padding field is used to ensure that the data begins on 32 bit word boundary. The padding is zero.
- Protocol In this document, the next higher level protocol identifier, an internet header field.
- Rest The local address portion of an Internet Address.
- Source The source address, an internet header field.
- TCP *Transmission Control Protocol*: A host-to-host protocol for reliable communication in internet environments.
- TCP Segment The unit of data exchanged between TCP modules (including the TCP header).
- TFTP *Trivial File Transfer Protocol*: A simple file transfer protocol built on UDP.
- Time to Live An internet header field which indicates the upper bound on how long this internet datagram may exist.
- TOS Type of Service
- Total Length The internet header field Total Length is the length of the datagram in octets including internet header and data.
- TTL Time to Live

September 1981

Internet Protocol
Glossary

Type of Service

An internet header field which indicates the type (or quality) of service for this internet datagram.

UDP

User Datagram Protocol: A user level protocol for transaction oriented applications.

User

The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program.

Version

The Version field indicates the format of the internet header.

REFERENCES

- [1] Cerf, V., "The Catenet Model for Internetworking," Information Processing Techniques Office, Defense Advanced Research Projects Agency, IEN 48, July 1978.
- [2] Bolt Beranek and Newman, "Specification for the Interconnection of a Host and an IMP," BBN Technical Report 1822, Revised May 1978.
- [3] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," RFC 792, USC/Information Sciences Institute, September 1981.
- [4] Shoch, J., "Inter-Network Naming, Addressing, and Routing," COMPCON, IEEE Computer Society, Fall 1978.
- [5] Postel, J., "Address Mappings," RFC 796, USC/Information Sciences Institute, September 1981.
- [6] Shoch, J., "Packet Fragmentation in Inter-Network Protocols," Computer Networks, v. 3, n. 1, February 1979.
- [7] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.
- [8] Postel, J., "Service Mappings," RFC 795, USC/Information Sciences Institute, September 1981.
- [9] Postel, J., "Assigned Numbers," RFC 790, USC/Information Sciences Institute, September 1981.

Network Working Group
Request for Comments: 792

J. Postel
ISI
September 1981

Updates: RFCs 777, 760
Updates: IENs 109, 128

INTERNET CONTROL MESSAGE PROTOCOL

DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION

Introduction

The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP) [3,4]. Occasionally a gateway or destination host will communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route.

The Internet Protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a datagram will be delivered or a control message will be returned. Some datagrams may still be undelivered without any report of their loss. The higher level protocols that use IP must implement their own reliability procedures if reliable communication is required.

The ICMP messages typically report errors in the processing of datagrams. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages. Also ICMP messages are only sent about errors in handling fragment zero of fragmented datagrams. (Fragment zero has the fragment offset equal zero).

Message Formats

ICMP messages are sent using the basic IP header. The first octet of the data portion of the datagram is a ICMP type field; the value of this field determines the format of the remaining data. Any field labeled "unused" is reserved for later extensions and must be zero when sent, but receivers should not use these fields (except to include them in the checksum). Unless otherwise noted under the individual format descriptions, the values of the internet header fields are as follows:

Version

4

IHL

Internet header length in 32-bit words.

Type of Service

0

Total Length

Length of internet header and data in octets.

Identification, Flags, Fragment Offset

Used in fragmentation, see [1].

Time to Live

Time to live in seconds; as this field is decremented at each machine in which the datagram is processed, the value in this field should be at least as great as the number of gateways which this datagram will traverse.

Protocol

ICMP = 1

Header Checksum

The 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

September 1981
RFC 792

Source Address

The address of the gateway or host that composes the ICMP message.
Unless otherwise noted, this can be any of a gateway's addresses.

Destination Address

The address of the gateway or host to which the message should be sent.

September 1981
RFC 792

datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

If, according to the information in the gateway's routing tables, the network specified in the internet destination field of a datagram is unreachable, e.g., the distance to the network is infinity, the gateway may send a destination unreachable message to the internet source host of the datagram. In addition, in some networks, the gateway may be able to determine if the internet destination host is unreachable. Gateways in these networks may send destination unreachable messages to the source host when the destination host is unreachable.

If, in the destination host, the IP module cannot deliver the datagram because the indicated protocol module or process port is not active, the destination host may send a destination unreachable message to the source host.

Another case is when a datagram must be fragmented to be forwarded by a gateway yet the Don't Fragment flag is on. In this case the gateway must discard the datagram and may return a destination unreachable message.

Codes 0, 1, 4, and 5 may be received from a gateway. Codes 2 and 3 may be received from a host.

September 1981
RFC 792

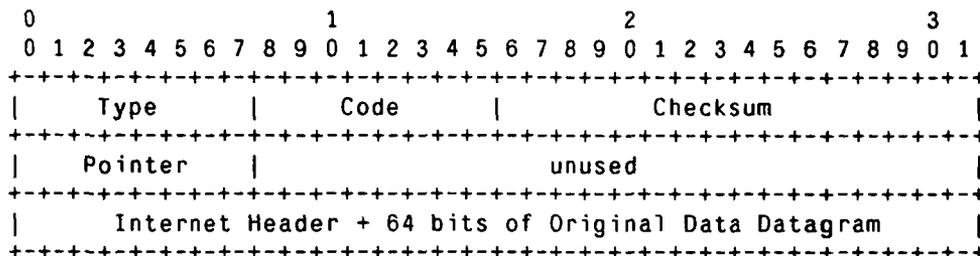
is zero it must discard the datagram. The gateway may also notify the source host via the time exceeded message.

If a host reassembling a fragmented datagram cannot complete the reassembly due to missing fragments within its time limit it discards the datagram, and it may send a time exceeded message.

If fragment zero is not available then no time exceeded need be sent at all.

Code 0 may be received from a gateway. Code 1 may be received from a host.

Parameter Problem Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

12

Code

0 = pointer indicates the error.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Pointer

If code = 0, identifies the octet where an error was detected.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

September 1981
RFC 792

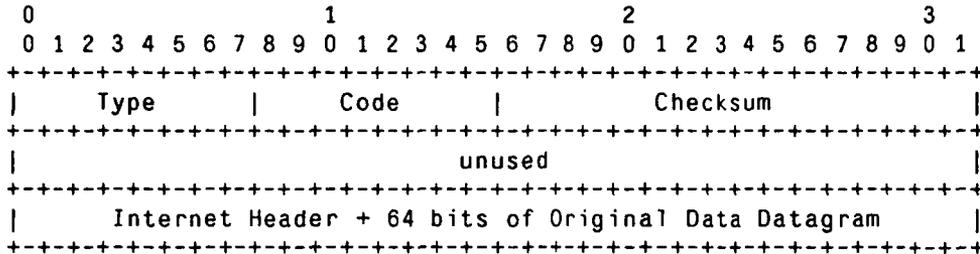
Description

If the gateway or host processing a datagram finds a problem with the header parameters such that it cannot complete processing the datagram it must discard the datagram. One potential source of such a problem is with incorrect arguments in an option. The gateway or host may also notify the source host via the parameter problem message. This message is only sent if the error caused the datagram to be discarded.

The pointer identifies the octet of the original datagram's header where the error was detected (it may be in the middle of an option). For example, 1 indicates something is wrong with the Type of Service, and (if there are options present) 20 indicates something is wrong with the type code of the first option.

Code 0 may be received from a gateway or a host.

Source Quench Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

4

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

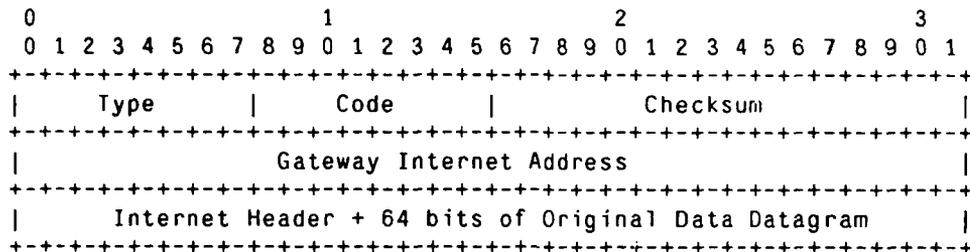
A gateway may discard internet datagrams if it does not have the buffer space needed to queue the datagrams for output to the next network on the route to the destination network. If a gateway

discards a datagram, it may send a source quench message to the internet source host of the datagram. A destination host may also send a source quench message if datagrams arrive too fast to be processed. The source quench message is a request to the host to cut back the rate at which it is sending traffic to the internet destination. The gateway may send a source quench message for every message that it discards. On receipt of a source quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages from the gateway. The source host can then gradually increase the rate at which it sends traffic to the destination until it again receives source quench messages.

The gateway or host may send the source quench message when it approaches its capacity limit rather than waiting until the capacity is exceeded. This means that the data datagram which triggered the source quench message may be delivered.

Code 0 may be received from a gateway or a host.

Redirect Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

5

Code

0 = Redirect datagrams for the Network.

1 = Redirect datagrams for the Host.

2 = Redirect datagrams for the Type of Service and Network.

3 = Redirect datagrams for the Type of Service and Host.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Gateway Internet Address

Address of the gateway to which traffic for the network specified in the internet destination network field of the original datagram's data should be sent.

September 1981
792

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

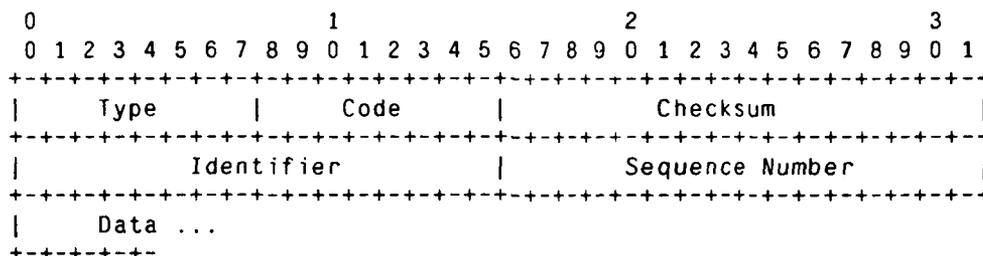
Description

The gateway sends a redirect message to a host in the following situation. A gateway, G1, receives an internet datagram from a host on a network to which the gateway is attached. The gateway, G1, checks its routing table and obtains the address of the next gateway, G2, on the route to the datagram's internet destination network, X. If G2 and the host identified by the internet source address of the datagram are on the same network, a redirect message is sent to the host. The redirect message advises the host to send its traffic for network X directly to gateway G2 as this is a shorter path to the destination. The gateway forwards the original datagram's data to its internet destination.

For datagrams with the IP source route options and the gateway address in the destination address field, a redirect message is not sent even if there is a better route to the ultimate destination than the next address in the source route.

Codes 0, 1, 2, and 3 may be received from a gateway.

Echo or Echo Reply Message



IP Fields:

Addresses

The address of the source in an echo message will be the destination of the echo reply message. To form an echo reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum recomputed.

IP Fields:

Type

- 8 for echo message;
- 0 for echo reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. If the total length is odd, the received data is padded with one octet of zeros for computing the checksum. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching echos and replies, may be zero.

Sequence Number

September 1981
RFC 792

If code = 0, a sequence number to aid in matching echos and replies, may be zero.

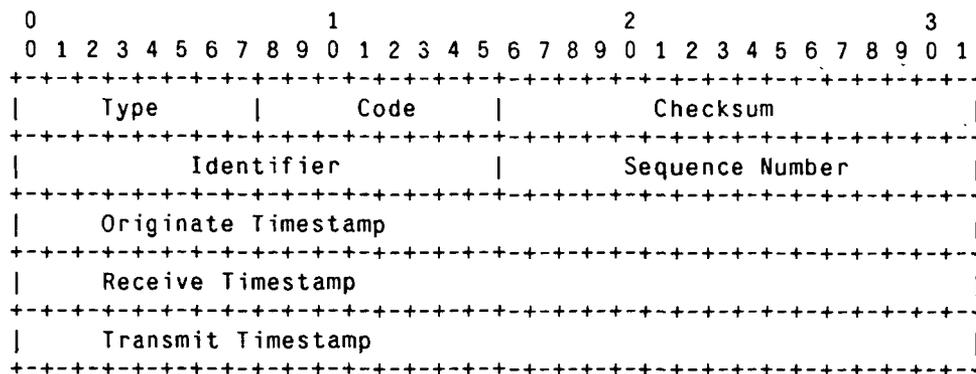
Description

The data received in the echo message must be returned in the echo reply message.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the echo requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply.

Code 0 may be received from a gateway or a host.

Timestamp or Timestamp Reply Message



IP Fields:

Addresses

The address of the source in a timestamp message will be the destination of the timestamp reply message. To form a timestamp reply message, the source and destination addresses are simply reversed, the type code changed to 14, and the checksum recomputed.

IP Fields:

Type

- 13 for timestamp message;
- 14 for timestamp reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Identifier

September 1981
RFC 792

If code = 0, an identifier to aid in matching timestamp and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching timestamp and replies, may be zero.

Description

The data received (a timestamp) in the message is returned in the reply together with an additional timestamp. The timestamp is 32 bits of milliseconds since midnight UT. One use of these timestamps is described by Mills [5].

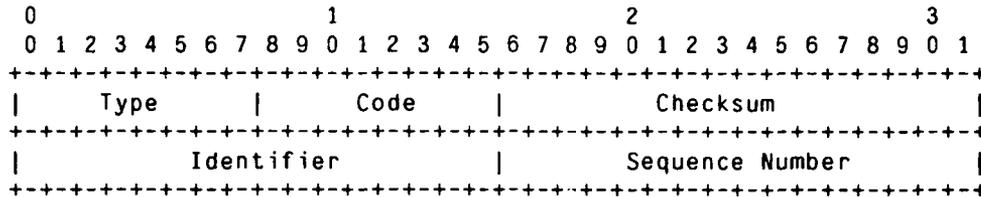
The Originate Timestamp is the time the sender last touched the message before sending it, the Receive Timestamp is the time the echoer first touched it on receipt, and the Transmit Timestamp is the time the echoer last touched the message on sending it.

If the time is not available in milliseconds or cannot be provided with respect to midnight UT then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

Information Request or Information Reply Message



IP Fields:

Addresses

The address of the source in a information request message will be the destination of the information reply message. To form a information reply message, the source and destination addresses are simply reversed, the type code changed to 16, and the checksum recomputed.

IP Fields:

Type

- 15 for information request message;
- 16 for information reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum , the checksum field should be zero. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching request and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching request and replies, may be zero.

September 1981
RFC 792

Description

This message may be sent with the source network in the IP header source and destination address fields zero (which means "this" network). The replying IP module should send the reply with the addresses fully specified. This message is a way for a host to find out the number of the network it is on.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

Summary of Message Types

- 0 Echo Reply
- 3 Destination Unreachable
- 4 Source Quench
- 5 Redirect
- 8 Echo
- 11 Time Exceeded
- 12 Parameter Problem
- 13 Timestamp
- 14 Timestamp Reply
- 15 Information Request
- 16 Information Reply

September 1981
RFC 792

References

- [1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.
- [2] Cerf, V., "The Catenet Model for Internetworking," IEN 48, Information Processing Techniques Office, Defense Advanced Research Projects Agency, July 1978.
- [3] Strazisar, V., "Gateway Routing: An Implementation Specification", IEN 30, Bolt Beranek and Newman, April 1979.
- [4] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.
- [5] Mills, D., "DCNET Internet Clock Service," RFC 778, COMSAT Laboratories, April 1981.

RFC: 793

TRANSMISSION CONTROL PROTOCOL

DARPA INTERNET PROGRAM

PROTOCOL SPECIFICATION

September 1981

prepared for

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

by

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

TABLE OF CONTENTS

PREFACE	iii
1. INTRODUCTION	1
1.1 Motivation	1
1.2 Scope	2
1.3 About This Document	2
1.4 Interfaces	3
1.5 Operation	3
2. PHILOSOPHY	7
2.1 Elements of the Internetwork System	7
2.2 Model of Operation	7
2.3 The Host Environment	8
2.4 Interfaces	9
2.5 Relation to Other Protocols	9
2.6 Reliable Communication	9
2.7 Connection Establishment and Clearing	10
2.8 Data Communication	12
2.9 Precedence and Security	13
2.10 Robustness Principle	13
3. FUNCTIONAL SPECIFICATION	15
3.1 Header Format	15
3.2 Terminology	19
3.3 Sequence Numbers	24
3.4 Establishing a connection	30
3.5 Closing a Connection	37
3.6 Precedence and Security	40
3.7 Data Communication	40
3.8 Interfaces	44
3.9 Event Processing	52
GLOSSARY	79
REFERENCES	85

September 1981

Transmission Control Protocol

PREFACE

This document describes the DoD Standard Transmission Control Protocol (TCP). There have been nine earlier editions of the ARPA TCP specification on which this standard is based, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition clarifies several details and removes the end-of-letter buffer-size adjustments, and redescribes the letter mechanism as a push function.

Jon Postel

Editor

RFC: 793
Replaces: RFC 761
IENs: 129, 124, 112, 81,
55, 44, 40, 27, 21, 5

TRANSMISSION CONTROL PROTOCOL

DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION

1. INTRODUCTION

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.

This document describes the functions to be performed by the Transmission Control Protocol, the program that implements it, and its interface to programs or users that require its services.

1.1. Motivation

Computer communication systems are playing an increasingly important role in military, government, and civilian environments. This document focuses its attention primarily on military computer communication requirements, especially robustness in the presence of communication unreliability and availability in the presence of congestion, but many of these problems are found in the civilian and government sector as well.

As strategic and tactical computer communication networks are developed and deployed, it is essential to provide means of interconnecting them and to provide standard interprocess communication protocols which can support a broad range of applications. In anticipation of the need for such standards, the Deputy Undersecretary of Defense for Research and Engineering has declared the Transmission Control Protocol (TCP) described herein to be a basis for DoD-wide inter-process communication protocol standardization.

TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. Very few assumptions are made as to the reliability of the communication protocols below the TCP layer. TCP assumes it can obtain a simple, potentially unreliable datagram service from the lower level protocols. In principle, the TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks.

Transmission Control Protocol Introduction

TCP is based on concepts first described by Cerf and Kahn in [1]. The TCP fits into a layered protocol architecture just above a basic Internet Protocol [2] which provides a way for the TCP to send and receive variable-length segments of information enclosed in internet datagram "envelopes". The internet datagram provides a means for addressing source and destination TCPs in different networks. The internet protocol also deals with any fragmentation or reassembly of the TCP segments required to achieve transport and delivery through multiple networks and interconnecting gateways. The internet protocol also carries information on the precedence, security classification and compartmentation of the TCP segments, so this information can be communicated end-to-end across multiple networks.

Protocol Layering

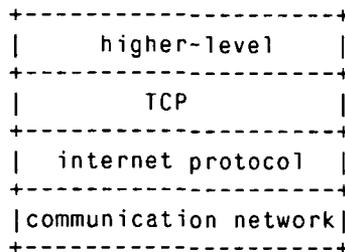


Figure 1

Much of this document is written in the context of TCP implementations which are co-resident with higher level protocols in the host computer. Some computer systems will be connected to networks via front-end computers which house the TCP and internet protocol layers, as well as network specific software. The TCP specification describes an interface to the higher level protocols which appears to be implementable even for the front-end case, as long as a suitable host-to-front end protocol is implemented.

1.2. Scope

The TCP is intended to provide a reliable process-to-process communication service in a multinet environment. The TCP is intended to be a host-to-host protocol in common use in multiple networks.

1.3. About this Document

This document represents a specification of the behavior required of any TCP implementation, both in its interactions with higher level protocols and in its interactions with other TCPs. The rest of this

section offers a very brief view of the protocol interfaces and operation. Section 2 summarizes the philosophical basis for the TCP design. Section 3 offers both a detailed description of the actions required of TCP when various events occur (arrival of new segments, user calls, errors, etc.) and the details of the formats of TCP segments.

1.4. Interfaces

The TCP interfaces on one side to user or application processes and on the other side to a lower level protocol such as Internet Protocol.

The interface between an application process and the TCP is illustrated in reasonable detail. This interface consists of a set of calls much like the calls an operating system provides to an application process for manipulating files. For example, there are calls to open and close connections and to send and receive data on established connections. It is also expected that the TCP can asynchronously communicate with application programs. Although considerable freedom is permitted to TCP implementors to design interfaces which are appropriate to a particular operating system environment, a minimum functionality is required at the TCP/user interface for any valid implementation.

The interface between TCP and lower level protocol is essentially unspecified except that it is assumed there is a mechanism whereby the two levels can asynchronously pass information to each other. Typically, one expects the lower level protocol to specify this interface. TCP is designed to work in a very general environment of interconnected networks. The lower level protocol which is assumed throughout this document is the Internet Protocol [2].

1.5. Operation

As noted above, the primary purpose of the TCP is to provide reliable, securable logical circuit or connection service between pairs of processes. To provide this service on top of a less reliable internet communication system requires facilities in the following areas:

- Basic Data Transfer
- Reliability
- Flow Control
- Multiplexing
- Connections
- Precedence and Security

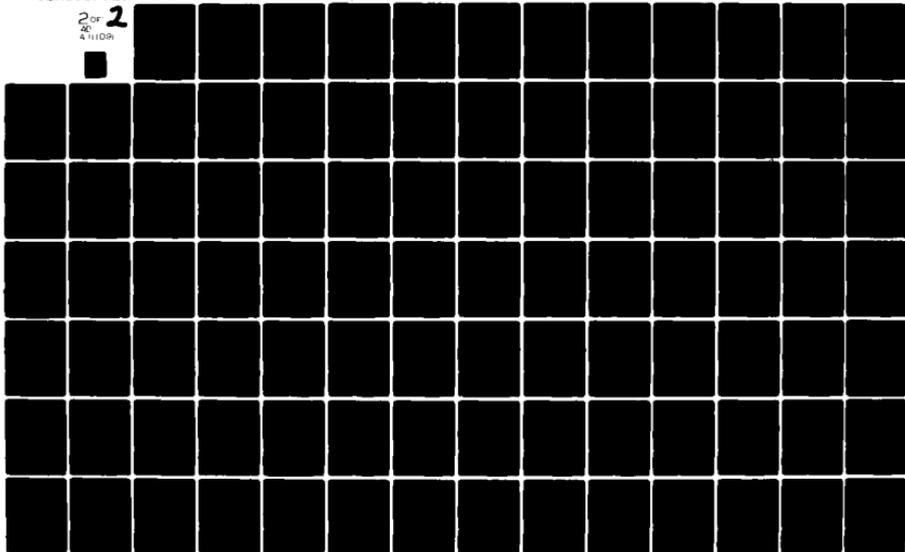
The basic operation of the TCP in each of these areas is described in the following paragraphs.

AD-A111 091

UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFO--ETC F/G 17/2
DARPA INTERNET PROGRAM. INTERNET AND TRANSMISSION CONTROL SPECI--ETC(U)
SEP 81 J B POSTEL MDA903-81-C-0336
ISI-RFC790-796 NL

UNCLASSIFIED

2 of 2
4110R



Transmission Control Protocol
Introduction

Basic Data Transfer:

The TCP is able to transfer a continuous stream of octets in each direction between its users by packaging some number of octets into segments for transmission through the internet system. In general, the TCPs decide when to block and forward data at their own convenience.

Sometimes users need to be sure that all the data they have submitted to the TCP has been transmitted. For this purpose a push function is defined. To assure that data submitted to a TCP is actually transmitted the sending user indicates that it should be pushed through to the receiving user. A push causes the TCPs to promptly forward and deliver data up to that point to the receiver. The exact push point might not be visible to the receiving user and the push function does not supply a record boundary marker.

Reliability:

The TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the internet communication system. This is achieved by assigning a sequence number to each octet transmitted, and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.

As long as the TCPs continue to function properly and the internet system does not become completely partitioned, no transmission errors will affect the correct delivery of data. TCP recovers from internet communication system errors.

Flow Control:

TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.

Multiplexing:

To allow for many processes within a single Host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a socket. A pair of sockets uniquely identifies each connection. That is, a socket may be simultaneously used in multiple connections.

The binding of ports to processes is handled independently by each Host. However, it proves useful to attach frequently used processes (e.g., a "logger" or timesharing service) to fixed sockets which are made known to the public. These services can then be accessed through the known addresses. Establishing and learning the port addresses of other processes may involve more dynamic mechanisms.

Connections:

The reliability and flow control mechanisms described above require that TCPs initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection. Each connection is uniquely specified by a pair of sockets identifying its two sides.

When two processes wish to communicate, their TCP's must first establish a connection (initialize the status information on each side). When their communication is complete, the connection is terminated or closed to free the resources for other uses.

Since connections must be established between unreliable hosts and over the unreliable internet communication system, a handshake mechanism with clock-based sequence numbers is used to avoid erroneous initialization of connections.

Precedence and Security:

The users of TCP may indicate the security and precedence of their communication. Provision is made for default values to be used when these features are not needed.

September 1981

Transmission Control Protocol

2. PHILOSOPHY

2.1. Elements of the Internetwork System

The internetwork environment consists of hosts connected to networks which are in turn interconnected via gateways. It is assumed here that the networks may be either local networks (e.g., the ETHERNET) or large networks (e.g., the ARPANET), but in any case are based on packet switching technology. The active agents that produce and consume messages are processes. Various levels of protocols in the networks, the gateways, and the hosts support an interprocess communication system that provides two-way data flow on logical connections between process ports.

The term packet is used generically here to mean the data of one transaction between a host and its network. The format of data blocks exchanged within the a network will generally not be of concern to us.

Hosts are computers attached to a network, and from the communication network's point of view, are the sources and destinations of packets. Processes are viewed as the active elements in host computers (in accordance with the fairly common definition of a process as a program in execution). Even terminals and files or other I/O devices are viewed as communicating with each other through the use of processes. Thus, all communication is viewed as inter-process communication.

Since a process may need to distinguish among several communication streams between itself and another process (or processes), we imagine that each process may have a number of ports through which it communicates with the ports of other processes.

2.2. Model of Operation

Processes transmit data by calling on the TCP and passing buffers of data as arguments. The TCP packages the data from these buffers into segments and calls on the internet module to transmit each segment to the destination TCP. The receiving TCP places the data from a segment into the receiving user's buffer and notifies the receiving user. The TCPs include control information in the segments which they use to ensure reliable ordered data transmission.

The model of internet communication is that there is an internet protocol module associated with each TCP which provides an interface to the local network. This internet module packages TCP segments inside internet datagrams and routes these datagrams to a destination internet module or intermediate gateway. To transmit the datagram through the local network, it is embedded in a local network packet.

The packet switches may perform further packaging, fragmentation, or

Transmission Control Protocol
Philosophy

other operations to achieve the delivery of the local packet to the destination internet module.

At a gateway between networks, the internet datagram is "unwrapped" from its local packet and examined to determine through which network the internet datagram should travel next. The internet datagram is then "wrapped" in a local packet suitable to the next network and routed to the next gateway, or to the final destination.

A gateway is permitted to break up an internet datagram into smaller internet datagram fragments if this is necessary for transmission through the next network. To do this, the gateway produces a set of internet datagrams; each carrying a fragment. Fragments may be further broken into smaller fragments at subsequent gateways. The internet datagram fragment format is designed so that the destination internet module can reassemble fragments into internet datagrams.

A destination internet module unwraps the segment from the datagram (after reassembling the datagram, if necessary) and passes it to the destination TCP.

This simple model of the operation glosses over many details. One important feature is the type of service. This provides information to the gateway (or internet module) to guide it in selecting the service parameters to be used in traversing the next network. Included in the type of service information is the precedence of the datagram. Datagrams may also carry security information to permit host and gateways that operate in multilevel secure environments to properly segregate datagrams for security considerations.

2.3. The Host Environment

The TCP is assumed to be a module in an operating system. The users access the TCP much like they would access the file system. The TCP may call on other operating system functions, for example, to manage data structures. The actual interface to the network is assumed to be controlled by a device driver module. The TCP does not call on the network device driver directly, but rather calls on the internet datagram protocol module which may in turn call on the device driver.

The mechanisms of TCP do not preclude implementation of the TCP in a front-end processor. However, in such an implementation, a host-to-front-end protocol must provide the functionality to support the type of TCP-user interface described in this document.

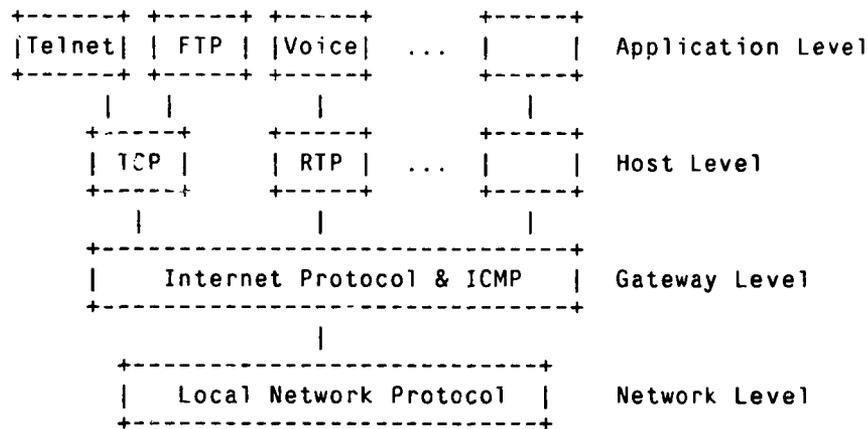
2.4. Interfaces

The TCP/user interface provides for calls made by the user on the TCP to OPEN or CLOSE a connection, to SEND or RECEIVE data, or to obtain STATUS about a connection. These calls are like other calls from user programs on the operating system, for example, the calls to open, read from, and close a file.

The TCP/internet interface provides calls to send and receive datagrams addressed to TCP modules in hosts anywhere in the internet system. These calls have parameters for passing the address, type of service, precedence, security, and other control information.

2.5. Relation to Other Protocols

The following diagram illustrates the place of the TCP in the protocol hierarchy:



Protocol Relationships

Figure 2.

It is expected that the TCP will be able to support higher level protocols efficiently. It should be easy to interface higher level protocols like the ARPANET Telnet or AUTODIN II THP to the TCP.

2.6. Reliable Communication

A stream of data sent on a TCP connection is delivered reliably and in order at the destination.

Transmission Control Protocol
Philosophy

Transmission is made reliable via the use of sequence numbers and acknowledgments. Conceptually, each octet of data is assigned a sequence number. The sequence number of the first octet of data in a segment is transmitted with that segment and is called the segment sequence number. Segments also carry an acknowledgment number which is the sequence number of the next expected data octet of transmissions in the reverse direction. When the TCP transmits a segment containing data, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that data is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted.

An acknowledgment by TCP does not guarantee that the data has been delivered to the end user, but only that the receiving TCP has taken the responsibility to do so.

To govern the flow of data between TCPs, a flow control mechanism is employed. The receiving TCP reports a "window" to the sending TCP. This window specifies the number of octets, starting with the acknowledgment number, that the receiving TCP is currently prepared to receive.

2.7. Connection Establishment and Clearing

To identify the separate data streams that a TCP may handle, the TCP provides a port identifier. Since port identifiers are selected independently by each TCP they might not be unique. To provide for unique addresses within each TCP, we concatenate an internet address identifying the TCP with a port identifier to create a socket which will be unique throughout all networks connected together.

A connection is fully specified by the pair of sockets at the ends. A local socket may participate in many connections to different foreign sockets. A connection can be used to carry data in both directions, that is, it is "full duplex".

TCPs are free to associate ports with processes however they choose. However, several basic concepts are necessary in any implementation. There must be well-known sockets which the TCP associates only with the "appropriate" processes by some means. We envision that processes may "own" ports, and that processes can initiate connections only on the ports they own. (Means for implementing ownership is a local issue, but we envision a Request Port user command, or a method of uniquely allocating a group of ports to a given process, e.g., by associating the high order bits of a port name with a given process.)

A connection is specified in the OPEN call by the local port and foreign socket arguments. In return, the TCP supplies a (short) local

connection name by which the user refers to the connection in subsequent calls. There are several things that must be remembered about a connection. To store this information we imagine that there is a data structure called a Transmission Control Block (TCB). One implementation strategy would have the local connection name be a pointer to the TCB for this connection. The OPEN call also specifies whether the connection establishment is to be actively pursued, or to be passively waited for.

A passive OPEN request means that the process wants to accept incoming connection requests rather than attempting to initiate a connection. Often the process requesting a passive OPEN will accept a connection request from any caller. In this case a foreign socket of all zeros is used to denote an unspecified socket. Unspecified foreign sockets are allowed only on passive OPENs.

A service process that wished to provide services for unknown other processes would issue a passive OPEN request with an unspecified foreign socket. Then a connection could be made with any process that requested a connection to this local socket. It would help if this local socket were known to be associated with this service.

Well-known sockets are a convenient mechanism for a priori associating a socket address with a standard service. For instance, the "Telnet-Server" process is permanently assigned to a particular socket, and other sockets are reserved for File Transfer, Remote Job Entry, Text Generator, Echoer, and Sink processes (the last three being for test purposes). A socket address might be reserved for access to a "Look-Up" service which would return the specific socket at which a newly created service would be provided. The concept of a well-known socket is part of the TCP specification, but the assignment of sockets to services is outside this specification. (See [4].)

Processes can issue passive OPENs and wait for matching active OPENs from other processes and be informed by the TCP when connections have been established. Two processes which issue active OPENs to each other at the same time will be correctly connected. This flexibility is critical for the support of distributed computing in which components act asynchronously with respect to each other.

There are two principal cases for matching the sockets in the local passive OPENs and an foreign active OPENs. In the first case, the local passive OPENs has fully specified the foreign socket. In this case, the match must be exact. In the second case, the local passive OPENs has left the foreign socket unspecified. In this case, any foreign socket is acceptable as long as the local sockets match. Other possibilities include partially restricted matches.

Transmission Control Protocol Philosophy

If there are several pending passive OPENs (recorded in TCBs) with the same local socket, an foreign active OPEN will be matched to a TCB with the specific foreign socket in the foreign active OPEN, if such a TCB exists, before selecting a TCB with an unspecified foreign socket.

The procedures to establish connections utilize the synchronize (SYN) control flag and involves an exchange of three messages. This exchange has been termed a three-way hand shake [3].

A connection is initiated by the rendezvous of an arriving segment containing a SYN and a waiting TCB entry each created by a user OPEN command. The matching of local and foreign sockets determines when a connection has been initiated. The connection becomes "established" when sequence numbers have been synchronized in both directions.

The clearing of a connection also involves the exchange of segments, in this case carrying the FIN control flag.

2.8. Data Communication

The data that flows on a connection may be thought of as a stream of octets. The sending user indicates in each SEND call whether the data in that call (and any preceeding calls) should be immediately pushed through to the receiving user by the setting of the PUSH flag.

A sending TCP is allowed to collect data from the sending user and to send that data in segments at its own convenience, until the push function is signaled, then it must send all unsent data. When a receiving TCP sees the PUSH flag, it must not wait for more data from the sending TCP before passing the data to the receiving process.

There is no necessary relationship between push functions and segment boundaries. The data in any particular segment may be the result of a single SEND call, in whole or part, or of multiple SEND calls.

The purpose of push function and the PUSH flag is to push data through from the sending user to the receiving user. It does not provide a record service.

There is a coupling between the push function and the use of buffers of data that cross the TCP/user interface. Each time a PUSH flag is associated with data placed into the receiving user's buffer, the buffer is returned to the user for processing even if the buffer is not filled. If data arrives that fills the user's buffer before a PUSH is seen, the data is passed to the user in buffer size units.

TCP also provides a means to communicate to the receiver of data that at some point further along in the data stream than the receiver is

September 1981

Transmission Control Protocol
Philosophy

currently reading there is urgent data. TCP does not attempt to define what the user specifically does upon being notified of pending urgent data, but the general notion is that the receiving process will take action to process the urgent data quickly.

2.9. Precedence and Security

The TCP makes use of the internet protocol type of service field and security option to provide precedence and security on a per connection basis to TCP users. Not all TCP modules will necessarily function in a multilevel secure environment; some may be limited to unclassified use only, and others may operate at only one security level and compartment. Consequently, some TCP implementations and services to users may be limited to a subset of the multilevel secure case.

TCP modules which operate in a multilevel secure environment must properly mark outgoing segments with the security, compartment, and precedence. Such TCP modules must also provide to their users or higher level protocols such as Telnet or THP an interface to allow them to specify the desired security level, compartment, and precedence of connections.

2.10. Robustness Principle

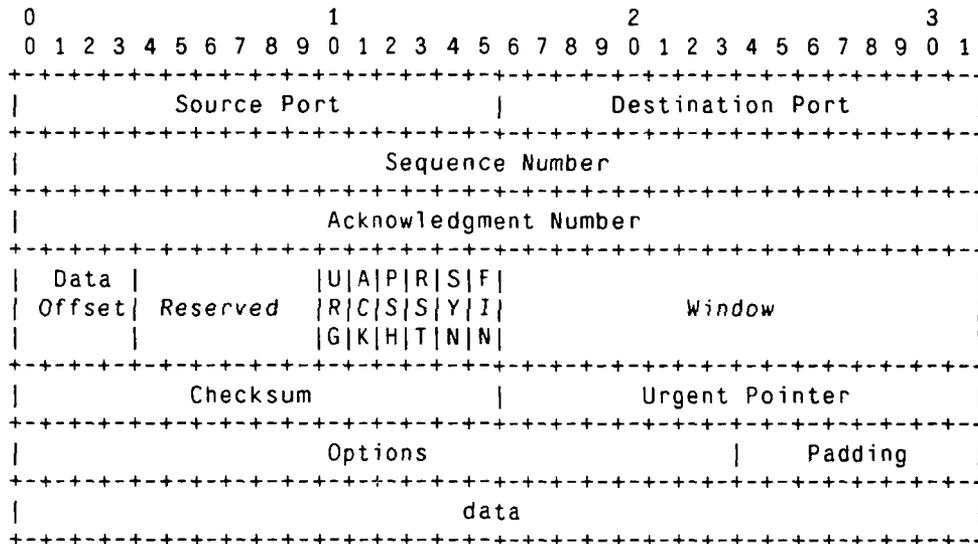
TCP implementations will follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.

3. FUNCTIONAL SPECIFICATION

3 1. Header Format

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses [2]. A TCP header follows the internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP.

TCP Header Format



TCP Header Format

Note that one tick mark represents one bit position.

Figure 3.

Source Port: 16 bits

The source port number.

Destination Port: 16 bits

The destination port number.

Transmission Control Protocol
Functional Specification

Sequence Number: 32 bits

The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number: 32 bits

If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Data Offset: 4 bits

The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

Reserved: 6 bits

Reserved for future use. Must be zero.

Control Bits: 6 bits (from left to right):

URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push Function
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender

Window: 16 bits

The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

Checksum: 16 bits

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers a 96 bit pseudo header conceptually

prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP.

```

+-----+-----+-----+-----+
|           Source Address           |
+-----+-----+-----+-----+
|           Destination Address      |
+-----+-----+-----+-----+
| zero | PTCL |   TCP Length   |
+-----+-----+-----+-----+

```

The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

Urgent Pointer: 16 bits

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.

Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets.

Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

A TCP must implement all options.

Transmission Control Protocol
Functional Specification

Currently defined options include (kind indicated in octal):

Kind	Length	Meaning
0	-	End of option list.
1	-	No-Operation.
2	4	Maximum Segment Size.

Specific Option Definitions

End of Option List

```
+-----+
|00000000|
+-----+
Kind=0
```

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation

```
+-----+
|00000001|
+-----+
Kind=1
```

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

Maximum Segment Size

```
+-----+-----+-----+-----+
|00000010|00000100| max seg size |
+-----+-----+-----+-----+
Kind=2 Length=4
```

Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

Padding: variable

The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

3.2. Terminology

Before we can discuss very much about the operation of the TCP we need to introduce some detailed terminology. The maintenance of a TCP connection requires the remembering of several variables. We conceive of these variables being stored in a connection record called a Transmission Control Block or TCB. Among the variables stored in the TCB are the local and remote socket numbers, the security and precedence of the connection, pointers to the user's send and receive buffers, pointers to the retransmit queue and to the current segment. In addition several variables relating to the send and receive sequence numbers are stored in the TCB.

Send Sequence Variables

SND.UNA - send unacknowledged
SND.NXT - send next
SND.WND - send window
SND.UP - send urgent pointer
SND.WL1 - segment sequence number used for last window update
SND.WL2 - segment acknowledgment number used for last window update
ISS - initial send sequence number

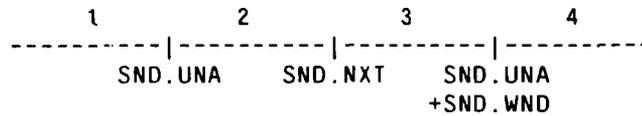
Receive Sequence Variables

RCV.NXT - receive next
RCV.WND - receive window
RCV.UP - receive urgent pointer
IRS - initial receive sequence number

Transmission Control Protocol
Functional Specification

The following diagrams may help to relate some of these variables to the sequence space.

Send Sequence Space



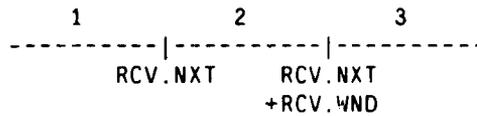
- 1 - old sequence numbers which have been acknowledged
- 2 - sequence numbers of unacknowledged data
- 3 - sequence numbers allowed for new data transmission
- 4 - future sequence numbers which are not yet allowed

Send Sequence Space

Figure 4.

The send window is the portion of the sequence space labeled 3 in figure 4.

Receive Sequence Space



- 1 - old sequence numbers which have been acknowledged
- 2 - sequence numbers allowed for new reception
- 3 - future sequence numbers which are not yet allowed

Receive Sequence Space

Figure 5.

The receive window is the portion of the sequence space labeled 2 in figure 5.

There are also some variables used frequently in the discussion that take their values from the fields of the current segment.

Current Segment Variables

SEG.SEQ - segment sequence number
SEG.ACK - segment acknowledgment number
SEG.LEN - segment length
SEG.WND - segment window
SEG.UP - segment urgent pointer
SEG.PRC - segment precedence value

A connection progresses through a series of states during its lifetime. The states are: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictional state CLOSED. CLOSED is fictional because it represents the state when there is no TCB, and therefore, no connection. Briefly the meanings of the states are:

LISTEN - represents waiting for a connection request from any remote TCP and port.

SYN-SENT - represents waiting for a matching connection request after having sent a connection request.

SYN-RECEIVED - represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

ESTABLISHED - represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

FIN-WAIT-1 - represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

FIN-WAIT-2 - represents waiting for a connection termination request from the remote TCP.

CLOSE-WAIT - represents waiting for a connection termination request from the local user.

CLOSING - represents waiting for a connection termination request acknowledgment from the remote TCP.

LAST-ACK - represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

September 1981

Transmission Control Protocol
Functional Specification

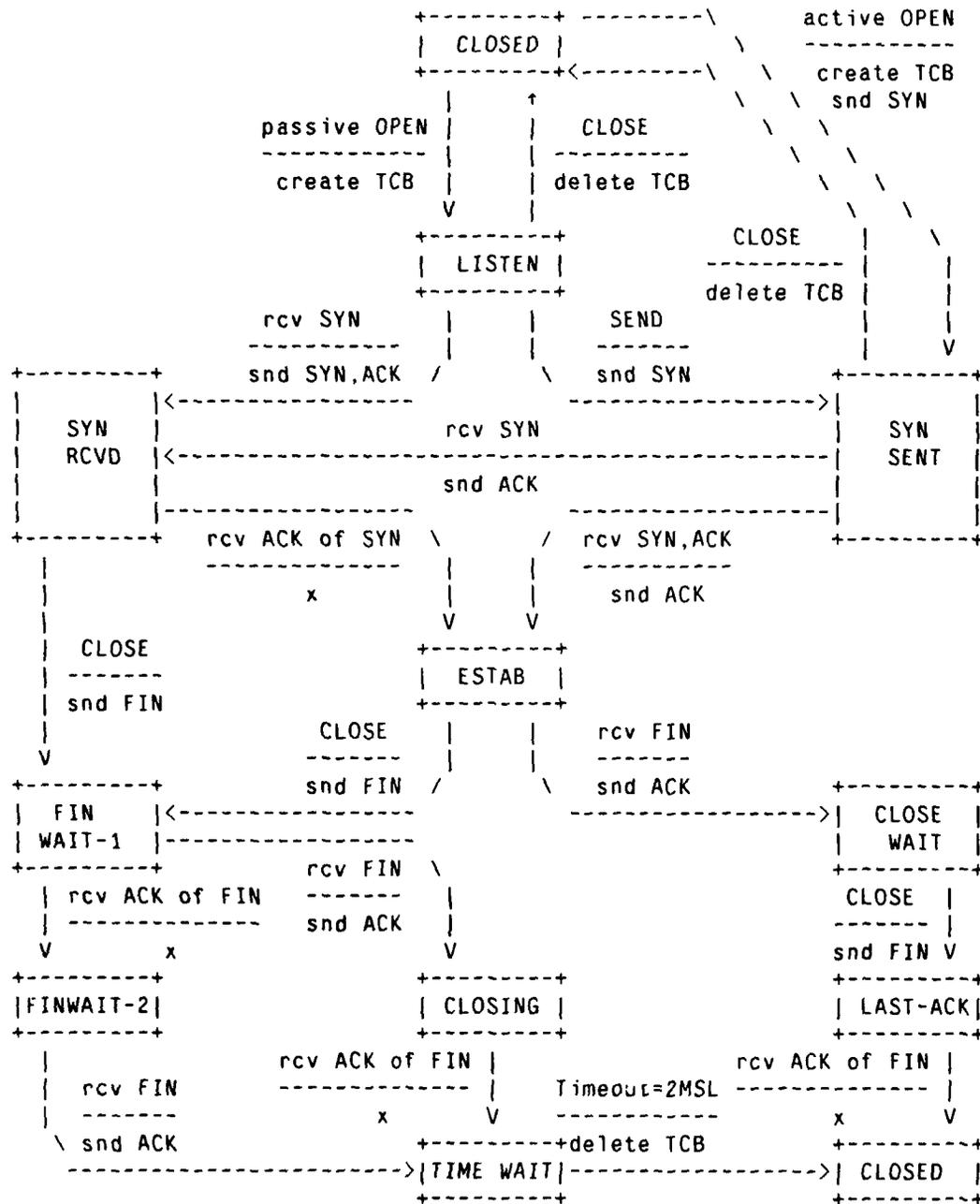
TIME-WAIT - represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request.

CLOSED - represents no connection state at all.

A TCP connection progresses from one state to another in response to events. The events are the user calls, OPEN, SEND, RECEIVE, CLOSE, ABORT, and STATUS; the incoming segments, particularly those containing the SYN, ACK, RST and FIN flags; and timeouts.

The state diagram in figure 6 illustrates only state changes, together with the causing events and resulting actions, but addresses neither error conditions nor actions which are not connected with state changes. In a later section, more detail is offered with respect to the reaction of the TCP to events.

NOTE BENE: this diagram is only a summary and must not be taken as the total specification.



TCP Connection State Diagram
Figure 6.

Transmission Control Protocol
Functional Specification

3.3. Sequence Numbers

A fundamental notion in the design is that every octet of data sent over a TCP connection has a sequence number. Since every octet is sequenced, each of them can be acknowledged. The acknowledgment mechanism employed is cumulative so that an acknowledgment of sequence number X indicates that all octets up to but not including X have been received. This mechanism allows for straight-forward duplicate detection in the presence of retransmission. Numbering of octets within a segment is that the first data octet immediately following the header is the lowest numbered, and the following octets are numbered consecutively.

It is essential to remember that the actual sequence number space is finite, though very large. This space ranges from 0 to $2^{32} - 1$. Since the space is finite, all arithmetic dealing with sequence numbers must be performed modulo 2^{32} . This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from $2^{32} - 1$ to 0 again. There are some subtleties to computer modulo arithmetic, so great care should be taken in programming the comparison of such values. The symbol " $=<$ " means "less than or equal" (modulo 2^{32}).

The typical kinds of sequence number comparisons which the TCP must perform include:

- (a) Determining that an acknowledgment refers to some sequence number sent but not yet acknowledged.
- (b) Determining that all sequence numbers occupied by a segment have been acknowledged (e.g., to remove the segment from a retransmission queue).
- (c) Determining that an incoming segment contains sequence numbers which are expected (i.e., that the segment "overlaps" the receive window).

September 1981

Transmission Control Protocol
Functional Specification

In response to sending data the TCP will receive acknowledgments. The following comparisons are needed to process the acknowledgments.

SND.UNA = oldest unacknowledged sequence number

SND.NXT = next sequence number to be sent

SEG.ACK = acknowledgment from the receiving TCP (next sequence number expected by the receiving TCP)

SEG.SEQ = first sequence number of a segment

SEG.LEN = the number of octets occupied by the data in the segment (counting SYN and FIN)

SEG.SEQ+SEG.LEN-1 = last sequence number of a segment

A new acknowledgment (called an "acceptable ack"), is one for which the inequality below holds:

$SND.UNA < SEG.ACK \leq SND.NXT$

A segment on the retransmission queue is fully acknowledged if the sum of its sequence number and length is less or equal than the acknowledgment value in the incoming segment.

When data is received the following comparisons are needed:

RCV.NXT = next sequence number expected on an incoming segments, and is the left or lower edge of the receive window

RCV.NXT+RCV.WND-1 = last sequence number expected on an incoming segment, and is the right or upper edge of the receive window

SEG.SEQ = first sequence number occupied by the incoming segment

SEG.SEQ+SEG.LEN-1 = last sequence number occupied by the incoming segment

A segment is judged to occupy a portion of valid receive sequence space if

$RCV.NXT \leq SEG.SEQ < RCV.NXT+RCV.WND$

or

$RCV.NXT \leq SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND$

Transmission Control Protocol
Functional Specification

The first part of this test checks to see if the beginning of the segment falls in the window, the second part of the test checks to see if the end of the segment falls in the window; if the segment passes either part of the test it contains data in the window.

Actually, it is a little more complicated than this. Due to zero windows and zero length segments, we have four cases for the acceptability of an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

Note that when the receive window is zero no segments should be acceptable except ACK segments. Thus, it is possible for a TCP to maintain a zero receive window while transmitting data and receiving ACKs. However, even when the receive window is zero, a TCP must process the RST and URG fields of all incoming segments.

We have taken advantage of the numbering scheme to protect certain control information as well. This is achieved by implicitly including some control flags in the sequence space so they can be retransmitted and acknowledged without confusion (i.e., one and only one copy of the control will be acted upon). Control information is not physically carried in the segment data space. Consequently, we must adopt rules for implicitly assigning sequence numbers to control. The SYN and FIN are the only controls requiring this protection, and these controls are used only at connection opening and closing. For sequence number purposes, the SYN is considered to occur before the first actual data octet of the segment in which it occurs, while the FIN is considered to occur after the last actual data octet in a segment in which it occurs. The segment length (SEG.LEN) includes both data and sequence space occupying controls. When a SYN is present then SEG.SEQ is the sequence number of the SYN.

Initial Sequence Number Selection

The protocol places no restriction on a particular connection being used over and over again. A connection is defined by a pair of sockets. New instances of a connection will be referred to as incarnations of the connection. The problem that arises from this is -- "how does the TCP identify duplicate segments from previous incarnations of the connection?" This problem becomes apparent if the connection is being opened and closed in quick succession, or if the connection breaks with loss of memory and is then reestablished.

To avoid confusion we must prevent segments from one incarnation of a connection from being used while the same sequence numbers may still be present in the network from an earlier incarnation. We want to assure this, even if a TCP crashes and loses all knowledge of the sequence numbers it has been using. When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32 bit ISN. The generator is bound to a (possibly fictitious) 32 bit clock whose low order bit is incremented roughly every 4 microseconds. Thus, the ISN cycles approximately every 4.55 hours. Since we assume that segments will stay in the network no more than the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55 hours we can reasonably assume that ISN's will be unique.

For each connection there is a send sequence number and a receive sequence number. The initial send sequence number (ISS) is chosen by the data sending TCP, and the initial receive sequence number (IRS) is learned during the connection establishing procedure.

For a connection to be established or initialized, the two TCPs must synchronize on each other's initial sequence numbers. This is done in an exchange of connection establishing segments carrying a control bit called "SYN" (for synchronize) and the initial sequence numbers. As a shorthand, segments carrying the SYN bit are also called "SYNs". Hence, the solution requires a suitable mechanism for picking an initial sequence number and a slightly involved handshake to exchange the ISN's.

The synchronization requires each side to send it's own initial sequence number and to receive a confirmation of it in acknowledgment from the other side. Each side must also receive the other side's initial sequence number and send a confirming acknowledgment.

- 1) A --> B SYN my sequence number is X
- 2) A <-- B ACK your sequence number is X
- 3) A <-- B SYN my sequence number is Y
- 4) A --> B ACK your sequence number is Y

September 1981

Transmission Control Protocol
Functional Specification

Because steps 2 and 3 can be combined in a single message this is called the three way (or three message) handshake.

A three way handshake is necessary because sequence numbers are not tied to a global clock in the network, and TCPs may have different mechanisms for picking the ISN's. The receiver of the first SYN has no way of knowing whether the segment was an old delayed one or not, unless it remembers the last sequence number used on the connection (which is not always possible), and so it must ask the sender to verify this SYN. The three way handshake and the advantages of a clock-driven scheme are discussed in [3].

Knowing When to Keep Quiet

To be sure that a TCP does not create a segment that carries a sequence number which may be duplicated by an old segment remaining in the network, the TCP must keep quiet for a maximum segment lifetime (MSL) before assigning any sequence numbers upon starting up or recovering from a crash in which memory of sequence numbers in use was lost. For this specification the MSL is taken to be 2 minutes. This is an engineering choice, and may be changed if experience indicates it is desirable to do so. Note that if a TCP is reinitialized in some sense, yet retains its memory of sequence numbers in use, then it need not wait at all; it must only be sure to use sequence numbers larger than those recently used.

The TCP Quiet Time Concept

This specification provides that hosts which "crash" without retaining any knowledge of the last sequence numbers transmitted on each active (i.e., not closed) connection shall delay emitting any TCP segments for at least the agreed Maximum Segment Lifetime (MSL) in the internet system of which the host is a part. In the paragraphs below, an explanation for this specification is given. TCP implementors may violate the "quiet time" restriction, but only at the risk of causing some old data to be accepted as new or new data rejected as old duplicated by some receivers in the internet system.

TCPs consume sequence number space each time a segment is formed and entered into the network output queue at a source host. The duplicate detection and sequencing algorithm in the TCP protocol relies on the unique binding of segment data to sequence space to the extent that sequence numbers will not cycle through all 2^{32} values before the segment data bound to those sequence numbers has been delivered and acknowledged by the receiver and all duplicate copies of the segments have "drained" from the internet. Without such an assumption, two distinct TCP segments could conceivably be

September 1981

Transmission Control Protocol
Functional Specification

assigned the same or overlapping sequence numbers, causing confusion at the receiver as to which data is new and which is old. Remember that each segment is bound to as many consecutive sequence numbers as there are octets of data in the segment.

Under normal conditions, TCPs keep track of the next sequence number to emit and the oldest awaiting acknowledgment so as to avoid mistakenly using a sequence number over before its first use has been acknowledged. This alone does not guarantee that old duplicate data is drained from the net, so the sequence space has been made very large to reduce the probability that a wandering duplicate will cause trouble upon arrival. At 2 megabits/sec. it takes 4.5 hours to use up 2^{32} octets of sequence space. Since the maximum segment lifetime in the net is not likely to exceed a few tens of seconds, this is deemed ample protection for foreseeable nets, even if data rates escalate to 10's of megabits/sec. At 100 megabits/sec, the cycle time is 5.4 minutes which may be a little short, but still within reason.

The basic duplicate detection and sequencing algorithm in TCP can be defeated, however, if a source TCP does not have any memory of the sequence numbers it last used on a given connection. For example, if the TCP were to start all connections with sequence number 0, then upon crashing and restarting, a TCP might re-form an earlier connection (possibly after half-open connection resolution) and emit packets with sequence numbers identical to or overlapping with packets still in the network which were emitted on an earlier incarnation of the same connection. In the absence of knowledge about the sequence numbers used on a particular connection, the TCP specification recommends that the source delay for MSL seconds before emitting segments on the connection, to allow time for segments from the earlier connection incarnation to drain from the system.

Even hosts which can remember the time of day and used it to select initial sequence number values are not immune from this problem (i.e., even if time of day is used to select an initial sequence number for each new connection incarnation).

Suppose, for example, that a connection is opened starting with sequence number S . Suppose that this connection is not used much and that eventually the initial sequence number function ($ISN(t)$) takes on a value equal to the sequence number, say S_1 , of the last segment sent by this TCP on a particular connection. Now suppose, at this instant, the host crashes, recovers, and establishes a new incarnation of the connection. The initial sequence number chosen is $S_1 = ISN(t)$ -- last used sequence number on old incarnation of connection! If the recovery occurs quickly enough, any old

Transmission Control Protocol
Functional Specification

duplicates in the net bearing sequence numbers in the neighborhood of S1 may arrive and be treated as new packets by the receiver of the new incarnation of the connection.

The problem is that the recovering host may not know for how long it crashed nor does it know whether there are still old duplicates in the system from earlier connection incarnations.

One way to deal with this problem is to deliberately delay emitting segments for one MSL after recovery from a crash- this is the "quite time" specification. Hosts which prefer to avoid waiting are willing to risk possible confusion of old and new packets at a given destination may choose not to wait for the "quite time". Implementors may provide TCP users with the ability to select on a connection by connection basis whether to wait after a crash, or may informally implement the "quite time" for all connections. Obviously, even where a user selects to "wait," this is not necessary after the host has been "up" for at least MSL seconds.

To summarize: every segment emitted occupies one or more sequence numbers in the sequence space, the numbers occupied by a segment are "busy" or "in use" until MSL seconds have passed, upon crashing a block of space-time is occupied by the octets of the last emitted segment, if a new connection is started too soon and uses any of the sequence numbers in the space-time footprint of the last segment of the previous connection incarnation, there is a potential sequence number overlap area which could cause confusion at the receiver.

3.4. Establishing a connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP doesn't deliver the data to the user until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake reduces the possibility of false connections. It is the

implementation of a trade-off between memory and messages to provide information for this checking.

The simplest three-way handshake is shown in figure 7 below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (-->) indicate departure of a TCP segment from TCP A to TCP B, or arrival of a segment at B from A. Left arrows (<--), indicate the reverse. Ellipsis (...) indicates a segment which is still in the network (delayed). An "XXX" indicates a segment which is lost or rejected. Comments appear in parentheses. TCP states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of each line). Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

TCP A		TCP B
1. CLOSED		LISTEN
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
3. ESTABLISHED	<-- <SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
4. ESTABLISHED	--> <SEQ=101><ACK=301><CTL=ACK>	--> ESTABLISHED
5. ESTABLISHED	--> <SEQ=101><ACK=301><CTL=ACK><DATA>	--> ESTABLISHED

Basic 3-Way Handshake for Connection Synchronization

Figure 7.

In line 2 of figure 7, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100.

At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!).

Transmission Control Protocol
Functional Specification

Simultaneous initiation is only slightly more complex, as is shown in figure 8. Each TCP cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED.

TCP A		TCP B
1. CLOSED		CLOSED
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	...
3. SYN-RECEIVED	<-- <SEQ=300><CTL=SYN>	<-- SYN-SENT
4.	... <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
5. SYN-RECEIVED	--> <SEQ=100><ACK=301><CTL=SYN,ACK>	...
6. ESTABLISHED	<-- <SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
7.	... <SEQ=101><ACK=301><CTL=ACK>	--> ESTABLISHED

Simultaneous Connection Synchronization

Figure 8.

The principle reason for the three-way handshake is to prevent old duplicate connection initiations from causing confusion. To deal with this, a special control message, reset, has been devised. If the receiving TCP is in a non-synchronized state (i.e., SYN-SENT, SYN-RECEIVED), it returns to LISTEN on receiving an acceptable reset. If the TCP is in one of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it aborts the connection and informs its user. We discuss this latter case under "half-open" connections below.

TCP A		TCP B
1. CLOSED		LISTEN
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	...
3. (duplicate)	... <SEQ=90><CTL=SYN>	--> SYN-RECEIVED
4. SYN-SENT	<-- <SEQ=300><ACK=91><CTL=SYN,ACK>	<-- SYN-RECEIVED
5. SYN-SENT	--> <SEQ=91><CTL=RST>	--> LISTEN
6.	... <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
7. SYN-SENT	<-- <SEQ=400><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
8. ESTABLISHED	--> <SEQ=101><ACK=401><CTL=ACK>	--> ESTABLISHED

Recovery from Old Duplicate SYN

Figure 9.

As a simple example of recovery from old duplicates, consider figure 9. At line 3, an old duplicate SYN arrives at TCP B. TCP B cannot tell that this is an old duplicate, so it responds normally (line 4). TCP A detects that the ACK field is incorrect and returns a RST (reset) with its SEQ field selected to make the segment believable. TCP B, on receiving the RST, returns to the LISTEN state. When the original SYN (pun intended) finally arrives at line 6, the synchronization proceeds normally. If the SYN at line 6 had arrived before the RST, a more complex exchange might have occurred with RST's sent in both directions.

Half-Open Connections and Other Anomalies

An established connection is said to be "half-open" if one of the TCPs has closed or aborted the connection at its end without the knowledge of the other, or if the two ends of the connection have become desynchronized owing to a crash that resulted in loss of memory. Such connections will automatically become reset if an attempt is made to send data in either direction. However, half-open connections are expected to be unusual, and the recovery procedure is mildly involved.

If at site A the connection no longer exists, then an attempt by the

Transmission Control Protocol
Functional Specification

user at site B to send any data on it will result in the site B TCP receiving a reset control message. Such a message indicates to the site B TCP that something is wrong, and it is expected to abort the connection.

Assume that two user processes A and B are communicating with one another when a crash occurs causing loss of memory to A's TCP. Depending on the operating system supporting A's TCP, it is likely that some error recovery mechanism exists. When the TCP is up again, A is likely to start again from the beginning or from a recovery point. As a result, A will probably try to OPEN the connection again or try to SEND on the connection it believes open. In the latter case, it receives the error message "connection not open" from the local (A's) TCP. In an attempt to establish the connection, A's TCP will send a segment containing SYN. This scenario leads to the example shown in figure 10. After TCP A crashes, the user attempts to re-open the connection. TCP B, in the meantime, thinks the connection is open.

TCP A	TCP B
1. (CRASH)	(send 300, receive 100)
2. CLOSED	ESTABLISHED
3. SYN-SENT --> <SEQ=400><CTL=SYN>	--> (??)
4. (!!) <-- <SEQ=300><ACK=100><CTL=ACK>	<-- ESTABLISHED
5. SYN-SENT --> <SEQ=100><CTL=RST>	--> (Abort!!)
6. SYN-SENT	CLOSED
7. SYN-SENT --> <SEQ=400><CTL=SYN>	-->

Half-Open Connection Discovery

Figure 10.

When the SYN arrives at line 3, TCP B, being in a synchronized state, and the incoming segment outside the window, responds with an acknowledgment indicating what sequence it next expects to hear (ACK 100). TCP A sees that this segment does not acknowledge anything it sent and, being unsynchronized, sends a reset (RST) because it has detected a half-open connection. TCP B aborts at line 5. TCP A will

continue to try to establish the connection; the problem is now reduced to the basic 3-way handshake of figure 7.

An interesting alternative case occurs when TCP A crashes and TCP B tries to send data on what it thinks is a synchronized connection. This is illustrated in figure 11. In this case, the data arriving at TCP A from TCP B (line 2) is unacceptable because no such connection exists, so TCP A sends a RST. The RST is acceptable so TCP B processes it and aborts the connection.

TCP A	TCP B
1. (CRASH)	(send 300, receive 100)
2. (??) <-- <SEQ=300><ACK=100><DATA=10><CTL=ACK>	<-- ESTABLISHED
3. --> <SEQ=100><CTL=RST>	--> (ABORT!!)

Active Side Causes Half-Open Connection Discovery

Figure 11.

In figure 12, we find the two TCPs A and B with passive connections waiting for SYN. An old duplicate arriving at TCP B (line 2) stirs B into action. A SYN-ACK is returned (line 3) and causes TCP A to generate a RST (the ACK in line 3 is not acceptable). TCP B accepts the reset and returns to its passive LISTEN state.

TCP A	TCP B
1. LISTEN	LISTEN
2. ... <SEQ=Z><CTL=SYN>	--> SYN-RECEIVED
3. (??) <-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK>	<-- SYN-RECEIVED
4. --> <SEQ=Z+1><CTL=RST>	--> (return to LISTEN!)
5. LISTEN	LISTEN

Old Duplicate SYN Initiates a Reset on two Passive Sockets

Figure 12.

Transmission Control Protocol
Functional Specification

A variety of other cases are possible, all of which are accounted for by the following rules for RST generation and processing.

Reset Generation

As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case.

There are three groups of states:

1. If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to a non-existent connection are rejected by this means.

If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the CLOSED state.

2. If the connection is in any non-synchronized state (LISTEN, SYN-SENT, SYN-RECEIVED), and the incoming segment acknowledges something not yet sent (the segment carries an unacceptable ACK), or if an incoming segment has a security level or compartment which does not exactly match the level and compartment requested for the connection, a reset is sent.

If our SYN has not been acknowledged and the precedence level of the incoming segment is higher than the precedence level requested then either raise the local precedence level (if allowed by the user and the system) or send a reset; or if the precedence level of the incoming segment is lower than the precedence level requested then continue as if the precedence matched exactly (if the remote TCP cannot raise the precedence level to match ours this will be detected in the next segment it sends, and the connection will be terminated then). If our SYN has been acknowledged (perhaps in this incoming segment) the precedence level of the incoming segment must match the local precedence level exactly, if it does not a reset must be sent.

If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the same state.

3. If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), any unacceptable segment (out of window sequence number or unacceptable acknowledgment number) must elicit only an empty acknowledgment segment containing the current send-sequence number and an acknowledgment indicating the next sequence number expected to be received, and the connection remains in the same state.

If an incoming segment has a security level, or compartment, or precedence which does not exactly match the level, and compartment, and precedence requested for the connection, a reset is sent and connection goes to the CLOSED state. The reset takes its sequence number from the ACK field of the incoming segment.

Reset Processing

In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields. A reset is valid if its sequence number is in the window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN.

The receiver of a RST first validates it, then changes state. If the receiver was in the LISTEN state, it ignores it. If the receiver was in SYN-RECEIVED state and had previously been in the LISTEN state, then the receiver returns to the LISTEN state, otherwise the receiver aborts the connection and goes to the CLOSED state. If the receiver was in any other state, it aborts the connection and advises the user and goes to the CLOSED state.

3.5. Closing a Connection

CLOSE is an operation meaning "I have no more data to send." The notion of closing a full-duplex connection is subject to ambiguous interpretation, of course, since it may not be obvious how to treat the receiving side of the connection. We have chosen to treat CLOSE in a simplex fashion. The user who CLOSEs may continue to RECEIVE until he is told that the other side has CLOSED also. Thus, a program could initiate several SENDs followed by a CLOSE, and then continue to RECEIVE until signaled that a RECEIVE failed because the other side has CLOSED. We assume that the TCP will signal a user, even if no RECEIVES are outstanding, that the other side has closed, so the user can terminate his side gracefully. A TCP will reliably deliver all buffers SENT before the connection was CLOSED so a user who expects no data in return need only wait to hear the connection was CLOSED successfully to know that all his data was received at the destination TCP. Users must keep reading connections they close for sending until the TCP says no more data.

Transmission Control Protocol
Functional Specification

There are essentially three cases:

- 1) The user initiates by telling the TCP to CLOSE the connection
- 2) The remote TCP initiates by sending a FIN control signal
- 3) Both users CLOSE simultaneously

Case 1: Local user initiates the close

In this case, a FIN segment can be constructed and placed on the outgoing segment queue. No further SENDs from the user will be accepted by the TCP, and it enters the FIN-WAIT-1 state. RECEIVES are allowed in this state. All segments preceding and including FIN will be retransmitted until acknowledged. When the other TCP has both acknowledged the FIN and sent a FIN of its own, the first TCP can ACK this FIN. Note that a TCP receiving a FIN will ACK but not send its own FIN until its user has CLOSED the connection also.

Case 2: TCP receives a FIN from the network

If an unsolicited FIN arrives from the network, the receiving TCP can ACK it and tell the user that the connection is closing. The user will respond with a CLOSE, upon which the TCP can send a FIN to the other TCP after sending any remaining data. The TCP then waits until its own FIN is acknowledged whereupon it deletes the connection. If an ACK is not forthcoming, after the user timeout the connection is aborted and the user is told.

Case 3: both users close simultaneously

A simultaneous CLOSE by users at both ends of a connection causes FIN segments to be exchanged. When all segments preceding the FINs have been processed and acknowledged, each TCP can ACK the FIN it has received. Both will, upon receiving these ACKs, delete the connection.

TCP A		TCP B
1. ESTABLISHED		ESTABLISHED
2. (Close)		
FIN-WAIT-1	--> <SEQ=100><ACK=300><CTL=FIN,ACK>	--> CLOSE-WAIT
3. FIN-WAIT-2	<-- <SEQ=300><ACK=101><CTL=ACK>	<-- CLOSE-WAIT
4. TIME-WAIT	<-- <SEQ=300><ACK=101><CTL=FIN,ACK>	(Close) <-- LAST-ACK
5. TIME-WAIT	--> <SEQ=101><ACK=301><CTL=ACK>	--> CLOSED
6. (2 MSL) CLOSED		

Normal Close Sequence

Figure 13.

TCP A		TCP B
1. ESTABLISHED		ESTABLISHED
2. (Close)		(Close)
FIN-WAIT-1	--> <SEQ=100><ACK=300><CTL=FIN,ACK>	... FIN-WAIT-1
	<-- <SEQ=300><ACK=100><CTL=FIN,ACK>	<--
	... <SEQ=100><ACK=300><CTL=FIN,ACK>	-->
3. CLOSING	--> <SEQ=101><ACK=301><CTL=ACK>	... CLOSING
	<-- <SEQ=301><ACK=101><CTL=ACK>	<--
	... <SEQ=101><ACK=301><CTL=ACK>	-->
4. TIME-WAIT (2 MSL) CLOSED		TIME-WAIT (2 MSL) CLOSED

Simultaneous Close Sequence

Figure 14.

Transmission Control Protocol
Functional Specification

3.6. Precedence and Security

The intent is that connection be allowed only between ports operating with exactly the same security and compartment values and at the higher of the precedence level requested by the two ports.

The precedence and security parameters used in TCP are exactly those defined in the Internet Protocol (IP). Throughout this TCP specification the term "security/compartment" is intended to indicate the security parameters used in IP including security, compartment, user group, and handling restriction.

A connection attempt with mismatched security/compartment values or a lower precedence value must be rejected by sending a reset. Rejecting a connection due to too low a precedence only occurs after an acknowledgment of the SYN has been received.

Note that TCP modules which operate only at the default value of precedence will still have to check the precedence of incoming segments and possibly raise the precedence level they use on the connection.

The security parameters may be used even in a non-secure environment (the values would indicate unclassified data), thus hosts in non-secure environments must be prepared to receive the security parameters, though they need not send them.

3.7. Data Communication

Once the connection is established data is communicated by the exchange of segments. Because segments may be lost due to errors (checksum test failure), or network congestion, TCP uses retransmission (after a timeout) to ensure delivery of every segment. Duplicate segments may arrive due to network or TCP retransmission. As discussed in the section on sequence numbers the TCP performs certain tests on the sequence and acknowledgment numbers in the segments to verify their acceptability.

The sender of data keeps track of the next sequence number to use in the variable SND.NXT. The receiver of data keeps track of the next sequence number to expect in the variable RCV.NXT. The sender of data keeps track of the oldest unacknowledged sequence number in the variable SND.UNA. If the data flow is momentarily idle and all data sent has been acknowledged then the three variables will be equal.

When the sender creates a segment and transmits it the sender advances SND.NXT. When the receiver accepts a segment it advances RCV.NXT and sends an acknowledgment. When the data sender receives an

September 1981

Transmission Control Protocol
Functional Specification

acknowledgment it advances SND.UNA. The extent to which the values of these variables differ is a measure of the delay in the communication. The amount by which the variables are advanced is the length of the data in the segment. Note that once in the ESTABLISHED state all segments must carry current acknowledgment information.

The CLOSE user call implies a push function, as does the FIN control flag in an incoming segment.

Retransmission Timeout

Because of the variability of the networks that compose an internetwork system and the wide range of uses of TCP connections the retransmission timeout must be dynamically determined. One procedure for determining a retransmission time out is given here as an illustration.

An Example Retransmission Timeout Procedure

Measure the elapsed time between sending a data octet with a particular sequence number and receiving an acknowledgment that covers that sequence number (segments sent do not have to match segments received). This measured elapsed time is the Round Trip Time (RTT). Next compute a Smoothed Round Trip Time (SRTT) as:

$$\text{SRTT} = (\text{ALPHA} * \text{SRTT}) + ((1-\text{ALPHA}) * \text{RTT})$$

and based on this, compute the retransmission timeout (RTO) as:

$$\text{RTO} = \min[\text{UBOUND}, \max[\text{LBOUND}, (\text{BETA} * \text{SRTT})]]$$

where UBOUND is an upper bound on the timeout (e.g., 1 minute), LBOUND is a lower bound on the timeout (e.g., 1 second), ALPHA is a smoothing factor (e.g., .8 to .9), and BETA is a delay variance factor (e.g., 1.3 to 2.0).

The Communication of Urgent Information

The objective of the TCP urgent mechanism is to allow the sending user to stimulate the receiving user to accept some urgent data and to permit the receiving TCP to indicate to the receiving user when all the currently known urgent data has been received by the user.

This mechanism permits a point in the data stream to be designated as the end of urgent information. Whenever this point is in advance of the receive sequence number (RCV.NXT) at the receiving TCP, that TCP must tell the user to go into "urgent mode"; when the receive sequence number catches up to the urgent pointer, the TCP must tell user to go

Transmission Control Protocol
Functional Specification

into "normal mode". If the urgent pointer is updated while the user is in "urgent mode", the update will be invisible to the user.

The method employs a urgent field which is carried in all segments transmitted. The URG control flag indicates that the urgent field is meaningful and must be added to the segment sequence number to yield the urgent pointer. The absence of this flag indicates that there is no urgent data outstanding.

To send an urgent indication the user must also send at least one data octet. If the sending user also indicates a push, timely delivery of the urgent information to the destination process is enhanced.

Managing the Window

The window sent in each segment indicates the range of sequence numbers the sender of the window (the data receiver) is currently prepared to accept. There is an assumption that this is related to the currently available data buffer space available for this connection.

Indicating a large window encourages transmissions. If more data arrives than can be accepted, it will be discarded. This will result in excessive retransmissions, adding unnecessarily to the load on the network and the TCPs. Indicating a small window may restrict the transmission of data to the point of introducing a round trip delay between each new segment transmitted.

The mechanisms provided allow a TCP to advertise a large window and to subsequently advertise a much smaller window without having accepted that much data. This, so called "shrinking the window," is strongly discouraged. The robustness principle dictates that TCPs will not shrink the window themselves, but will be prepared for such behavior on the part of other TCPs.

The sending TCP must be prepared to accept from the user and send at least one octet of new data even if the send window is zero. The sending TCP must regularly retransmit to the receiving TCP even when the window is zero. Two minutes is recommended for the retransmission interval when the window is zero. This retransmission is essential to guarantee that when either TCP has a zero window the re-opening of the window will be reliably reported to the other.

When the receiving TCP has a zero window and a segment arrives it must still send an acknowledgment showing its next expected sequence number and current window (zero).

The sending TCP packages the data to be transmitted into segments

September 1981

Transmission Control Protocol
functional Specification

which fit the current window, and may repackage segments on the retransmission queue. Such repackaging is not required, but may be helpful.

In a connection with a one-way data flow, the window information will be carried in acknowledgment segments that all have the same sequence number so there will be no way to reorder them if they arrive out of order. This is not a serious problem, but it will allow the window information to be on occasion temporarily based on old reports from the data receiver. A refinement to avoid this problem is to act on the window information from segments that carry the highest acknowledgment number (that is segments with acknowledgment number equal or greater than the highest previously received).

The window management procedure has significant influence on the communication performance. The following comments are suggestions to implementers.

Window Management Suggestions

Allocating a very small window causes data to be transmitted in many small segments when better performance is achieved using fewer large segments.

One suggestion for avoiding small windows is for the receiver to defer updating a window until the additional allocation is at least X percent of the maximum allocation possible for the connection (where X might be 20 to 40).

Another suggestion is for the sender to avoid sending small segments by waiting until the window is large enough before sending data. If the user signals a push function then the data must be sent even if it is a small segment.

Note that the acknowledgments should not be delayed or unnecessary retransmissions will result. One strategy would be to send an acknowledgment when a small segment arrives (with out updating the window information), and then to send another acknowledgment with new window information when the window is larger.

The segment sent to probe a zero window may also begin a break up of transmitted data into smaller and smaller segments. If a segment containing a single data octet sent to probe a zero window is accepted, it consumes one octet of the window now available. If the sending TCP simply sends as much as it can whenever the window is non zero, the transmitted data will be broken into alternating big and small segments. As time goes on, occasional pauses in the receiver making window allocation available will

September 1981

Transmission Control Protocol
Functional Specification

result in breaking the big segments into a small and not quite so big pair. And after a while the data transmission will be in mostly small segments.

The suggestion here is that the TCP implementations need to actively attempt to combine small window allocations into larger windows, since the mechanisms for managing the window tend to lead to many small windows in the simplest minded implementations.

3.8. Interfaces

There are of course two interfaces of concern: the user/TCP interface and the TCP/lower-level interface. We have a fairly elaborate model of the user/TCP interface, but the interface to the lower level protocol module is left unspecified here, since it will be specified in detail by the specification of the lower level protocol. For the case that the lower level is IP we note some of the parameter values that TCPs might use.

User/TCP Interface

The following functional description of user commands to the TCP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different CP implementations may have different user interfaces. However, all TCPs must provide a certain minimum set of services to guarantee that all TCP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all TCP implementations.

TCP User Commands

The following sections functionally characterize a USER/TCP interface. The notation used is similar to most procedure or function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs).

The user commands described below specify the basic functions the TCP must perform to support interprocess communication. Individual implementations must define their own exact format, and may provide combinations or subsets of the basic functions in single calls. In particular, some implementations may wish to automatically OPEN a connection on the first SEND or RECEIVE issued by the user for a given connection.

In providing interprocess communication facilities, the TCP must not only accept commands, but must also return information to the processes it serves. The latter consists of:

- (a) general information about a connection (e.g., interrupts, remote close, binding of unspecified foreign socket).
- (b) replies to specific user commands indicating success or various types of failure.

Open

Format: OPEN (local port, foreign socket, active/passive
[, timeout] [, precedence] [, security/compartments] [, options])
-> local connection name

We assume that the local TCP is aware of the identity of the processes it serves and will check the authority of the process to use the connection specified. Depending upon the implementation of the TCP, the local network and TCP identifiers for the source address will either be supplied by the TCP or the lower level protocol (e.g., IP). These considerations are the result of concern about security, to the extent that no TCP be able to masquerade as another one, and so on. Similarly, no process can masquerade as another without the collusion of the TCP.

If the active/passive flag is set to passive, then this is a call to LISTEN for an incoming connection. A passive open may have either a fully specified foreign socket to wait for a particular connection or an unspecified foreign socket to wait for any call. A fully specified passive call can be made active by the subsequent execution of a SEND.

A transmission control block (TCB) is created and partially filled in with data from the OPEN command parameters.

On an active OPEN command, the TCP will begin the procedure to synchronize (i.e., establish) the connection at once.

The timeout, if present, permits the caller to set up a timeout for all data submitted to TCP. If data is not successfully delivered to the destination within the timeout period, the TCP will abort the connection. The present global default is five minutes.

The TCP or some component of the operating system will verify the users authority to open a connection with the specified

September 1981

Transmission Control Protocol
Functional Specification

precedence or security/compartement. The absence of precedence or security/compartement specification in the OPEN call indicates the default values must be used.

TCP will accept incoming requests as matching only if the security/compartement information is exactly the same and only if the precedence is equal to or higher than the precedence requested in the OPEN call.

The precedence for the connection is the higher of the values requested in the OPEN call and received from the incoming request, and fixed at that value for the life of the connection. Implementers may want to give the user control of this precedence negotiation. For example, the user might be allowed to specify that the precedence must be exactly matched, or that any attempt to raise the precedence be confirmed by the user.

A local connection name will be returned to the user by the TCP. The local connection name can then be used as a short hand term for the connection defined by the <local socket, foreign socket> pair.

Send

Format: SEND (local connection name, buffer address, byte count, PUSH flag, URGENT flag [,timeout])

This call causes the data contained in the indicated user buffer to be sent on the indicated connection. If the connection has not been opened, the SEND is considered an error. Some implementations may allow users to SEND first; in which case, an automatic OPEN would be done. If the calling process is not authorized to use this connection, an error is returned.

If the PUSH flag is set, the data must be transmitted promptly to the receiver, and the PUSH bit will be set in the last TCP segment created from the buffer. If the PUSH flag is not set, the data may be combined with data from subsequent SENDs for transmission efficiency.

If the URGENT flag is set, segments sent to the destination TCP will have the urgent pointer set. The receiving TCP will signal the urgent condition to the receiving process if the urgent pointer indicates that data preceding the urgent pointer has not been consumed by the receiving process. The purpose of urgent is to stimulate the receiver to process the urgent data and to indicate to the receiver when all the currently known urgent

data has been received. The number of times the sending user's TCP signals urgent will not necessarily be equal to the number of times the receiving user will be notified of the presence of urgent data.

If no foreign socket was specified in the OPEN, but the connection is established (e.g., because a LISTENing connection has become specific due to a foreign segment arriving for the local socket), then the designated buffer is sent to the implied foreign socket. Users who make use of OPEN with an unspecified foreign socket can make use of SEND without ever explicitly knowing the foreign socket address.

However, if a SEND is attempted before the foreign socket becomes specified, an error will be returned. Users can use the STATUS call to determine the status of the connection. In some implementations the TCP may notify the user when an unspecified socket is bound.

If a timeout is specified, the current user timeout for this connection is changed to the new one.

In the simplest implementation, SEND would not return control to the sending process until either the transmission was complete or the timeout had been exceeded. However, this simple method is both subject to deadlocks (for example, both sides of the connection might try to do SENDs before doing any RECEIVES) and offers poor performance, so it is not recommended. A more sophisticated implementation would return immediately to allow the process to run concurrently with network I/O, and, furthermore, to allow multiple SENDs to be in progress. Multiple SENDs are served in first come, first served order, so the TCP will queue those it cannot service immediately.

We have implicitly assumed an asynchronous user interface in which a SEND later elicits some kind of SIGNAL or pseudo-interrupt from the serving TCP. An alternative is to return a response immediately. For instance, SENDs might return immediate local acknowledgment, even if the segment sent had not been acknowledged by the distant TCP. We could optimistically assume eventual success. If we are wrong, the connection will close anyway due to the timeout. In implementations of this kind (synchronous), there will still be some asynchronous signals, but these will deal with the connection itself, and not with specific segments or buffers.

In order for the process to distinguish among error or success indications for different SENDs, it might be appropriate for the

Transmission Control Protocol
Functional Specification

buffer address to be returned along with the coded response to the SEND request. TCP-to-user signals are discussed below, indicating the information which should be returned to the calling process.

Receive

Format: RECEIVE (local connection name, buffer address, byte count) -> byte count, urgent flag, push flag

This command allocates a receiving buffer associated with the specified connection. If no OPEN precedes this command or the calling process is not authorized to use this connection, an error is returned.

In the simplest implementation, control would not return to the calling program until either the buffer was filled, or some error occurred, but this scheme is highly subject to deadlocks. A more sophisticated implementation would permit several RECEIVES to be outstanding at once. These would be filled as segments arrive. This strategy permits increased throughput at the cost of a more elaborate scheme (possibly asynchronous) to notify the calling program that a PUSH has been seen or a buffer filled.

If enough data arrive to fill the buffer before a PUSH is seen, the PUSH flag will not be set in the response to the RECEIVE. The buffer will be filled with as much data as it can hold. If a PUSH is seen before the buffer is filled the buffer will be returned partially filled and PUSH indicated.

If there is urgent data the user will have been informed as soon as it arrived via a TCP-to-user signal. The receiving user should thus be in "urgent mode". If the URGENT flag is on, additional urgent data remains. If the URGENT flag is off, this call to RECEIVE has returned all the urgent data, and the user may now leave "urgent mode". Note that data following the urgent pointer (non-urgent data) cannot be delivered to the user in the same buffer with preceding urgent data unless the boundary is clearly marked for the user.

To distinguish among several outstanding RECEIVES and to take care of the case that a buffer is not completely filled, the return code is accompanied by both a buffer pointer and a byte count indicating the actual length of the data received.

Alternative implementations of RECEIVE might have the TCP

September 1981

Transmission Control Protocol
Functional Specification

allocate buffer storage, or the TCP might share a ring buffer with the user.

Close

Format: CLOSE (local connection name)

This command causes the connection specified to be closed. If the connection is not open or the calling process is not authorized to use this connection, an error is returned. Closing connections is intended to be a graceful operation in the sense that outstanding SENDs will be transmitted (and retransmitted), as flow control permits, until all have been serviced. Thus, it should be acceptable to make several SEND calls, followed by a CLOSE, and expect all the data to be sent to the destination. It should also be clear that users should continue to RECEIVE on CLOSING connections, since the other side may be trying to transmit the last of its data. Thus, CLOSE means "I have no more to send" but does not mean "I will not receive any more." It may happen (if the user level protocol is not well thought out) that the closing side is unable to get rid of all its data before timing out. In this event, CLOSE turns into ABORT, and the closing TCP gives up.

The user may CLOSE the connection at any time on his own initiative, or in response to various prompts from the TCP (e.g., remote close executed, transmission timeout exceeded, destination inaccessible).

Because closing a connection requires communication with the foreign TCP, connections may remain in the closing state for a short time. Attempts to reopen the connection before the TCP replies to the CLOSE command will result in error responses.

Close also implies push function.

Status

Format: STATUS (local connection name) -> status data

This is an implementation dependent user command and could be excluded without adverse effect. Information returned would typically come from the TCB associated with the connection.

This command returns a data block containing the following information:

local socket,

Transmission Control Protocol
Functional Specification

foreign socket,
 local connection name,
 receive window,
 send window,
 connection state,
 number of buffers awaiting acknowledgment,
 number of buffers pending receipt,
 urgent state,
 precedence,
 security/compartments,
 and transmission timeout.

Depending on the state of the connection, or on the implementation itself, some of this information may not be available or meaningful. If the calling process is not authorized to use this connection, an error is returned. This prevents unauthorized processes from gaining information about a connection.

Abort

Format: ABORT (local connection name)

This command causes all pending SENDs and RECEIVES to be aborted, the TCB to be removed, and a special RESET message to be sent to the TCP on the other side of the connection. Depending on the implementation, users may receive abort indications for each outstanding SEND or RECEIVE, or may simply receive an ABORT-acknowledgment.

TCP-to-User Messages

It is assumed that the operating system environment provides a means for the TCP to asynchronously signal the user program. When the TCP does signal a user program, certain information is passed to the user. Often in the specification the information will be an error message. In other cases there will be information relating to the completion of processing a SEND or RECEIVE or other user call.

The following information is provided:

Local Connection Name	Always
Response String	Always
Buffer Address	Send & Receive
Byte count (counts bytes received)	Receive
Push flag	Receive
Urgent flag	Receive

September 1981

Transmission Control Protocol
Functional Specification

TCP/Lower-Level Interface

The TCP calls on a lower level protocol module to actually send and receive information over a network. One case is that of the ARPA internetwork system where the lower level module is the Internet Protocol (IP) [2].

If the lower level protocol is IP it provides arguments for a type of service and for a time to live. TCP uses the following settings for these parameters:

Type of Service = Precedence: routine, Delay: normal, Throughput: normal, Reliability: normal; or 00000000.

Time to Live = one minute, or 00111100.

Note that the assumed maximum segment lifetime is two minutes. Here we explicitly ask that a segment be destroyed if it cannot be delivered by the internet system within one minute.

If the lower level is IP (or other protocol that provides this feature) and source routing is used, the interface must allow the route information to be communicated. This is especially important so that the source and destination addresses used in the TCP checksum be the originating source and ultimate destination. It is also important to preserve the return route to answer connection requests.

Any lower level protocol will have to provide the source address, destination address, and protocol fields, and some way to determine the "TCP length", both to provide the functional equivalent service of IP and to be used in the TCP checksum.

Transmission Control Protocol
Functional Specification

3.9. Event Processing

The processing depicted in this section is an example of one possible implementation. Other implementations may have slightly different processing sequences, but they should differ from those in this section only in detail, not in substance.

The activity of the TCP can be characterized as responding to events. The events that occur can be cast into three categories: user calls, arriving segments, and timeouts. This section describes the processing the TCP does in response to each of the events. In many cases the processing required depends on the state of the connection.

Events that occur:

User Calls

OPEN
SEND
RECEIVE
CLOSE
ABORT
STATUS

Arriving Segments

SEGMENT ARRIVES

Timeouts

USER TIMEOUT
RETRANSMISSION TIMEOUT
TIME-WAIT TIMEOUT

The model of the TCP/user interface is that user commands receive an immediate return and possibly a delayed response via an event or pseudo interrupt. In the following descriptions, the term "signal" means cause a delayed response.

Error responses are given as character strings. For example, user commands referencing connections that do not exist receive "error: connection not open".

Please note in the following that all arithmetic on sequence numbers, acknowledgment numbers, windows, et cetera, is modulo 2^{32} the size of the sequence number space. Also note that " $=<$ " means less than or equal to (modulo 2^{32}).

September 1981

Transmission Control Protocol
Functional Specification

A natural way to think about processing incoming segments is to imagine that they are first tested for proper sequence number (i.e., that their contents lie in the range of the expected "receive window" in the sequence number space) and then that they are generally queued and processed in sequence number order.

When a segment overlaps other already received segments we reconstruct the segment to contain just the new data, and adjust the header fields to be consistent.

Note that if no state change is mentioned the TCP stays in the same state.

OPEN Call

CLOSED STATE (i.e., TCB does not exist)

Create a new transmission control block (TCB) to hold connection state information. Fill in local socket identifier, foreign socket, precedence, security/compartments, and user timeout information. Note that some parts of the foreign socket may be unspecified in a passive OPEN and are to be filled in by the parameters of the incoming SYN segment. Verify the security and precedence requested are allowed for this user, if not return "error: precedence not allowed" or "error: security/compartments not allowed." If passive enter the LISTEN state and return. If active and the foreign socket is unspecified, return "error: foreign socket unspecified"; if active and the foreign socket is specified, issue a SYN segment. An initial send sequence number (ISS) is selected. A SYN segment of the form <SEQ=ISS><CTL=SYN> is sent. Set SND.UNA to ISS, SND.NXT to ISS+1, enter SYN-SENT state, and return.

If the caller does not have access to the local socket specified, return "error: connection illegal for this process". If there is no room to create a new connection, return "error: insufficient resources".

LISTEN STATE

If active and the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

September 1981

Transmission Control Protocol
Functional Specification

OPEN Call

SYN-SENT STATE
SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection already exists".

SEND Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, then return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

If the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

SYN-SENT STATE

SYN-RECEIVED STATE

Queue the data for transmission after entering ESTABLISHED state. If no space to queue, respond with "error: insufficient resources".

ESTABLISHED STATE

CLOSE-WAIT STATE

Segmentize the buffer and send it with a piggybacked acknowledgment (acknowledgment value = RCV.NXT). If there is insufficient space to remember this buffer, simply return "error: insufficient resources".

If the urgent flag is set, then $SND.UP \leftarrow SND.NXT - 1$ and set the urgent pointer in the outgoing segments.

September 1981

Transmission Control Protocol
Functional Specification

SEND Call

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection closing" and do not service request.

RECEIVE Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE
SYN-SENT STATE
SYN-RECEIVED STATE

Queue for processing after entering ESTABLISHED state. If there is no room to queue this request, respond with "error: insufficient resources".

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

If insufficient incoming segments are queued to satisfy the request, queue the request. If there is no queue space to remember the RECEIVE, respond with "error: insufficient resources".

Reassemble queued incoming segments into receive buffer and return to user. Mark "push seen" (PUSH) if this is the case.

If RCV.UP is in advance of the data currently being passed to the user notify the user of the presence of urgent data.

When the TCP takes responsibility for delivering data to the user that fact must be communicated to the sender via an acknowledgment. The formation of such an acknowledgment is described below in the discussion of processing an incoming segment.

September 1981

Transmission Control Protocol
Functional Specification

RECEIVE Call

CLOSE-WAIT STATE

Since the remote side has already sent FIN, RECEIVES must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection closing".

CLOSE Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVES are returned with "error: closing" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

Delete the TCB and return "error: closing" responses to any queued SENDS, or RECEIVES.

SYN-RECEIVED STATE

If no SENDS have been issued and there is no pending data to send, then form a FIN segment and send it, and enter FIN-WAIT-1 state; otherwise queue for processing after entering ESTABLISHED state.

ESTABLISHED STATE

Queue this until all preceding SENDS have been segmentized, then form a FIN segment and send it. In any case, enter FIN-WAIT-1 state.

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

Strictly speaking, this is an error and should receive a "error: connection closing" response. An "ok" response would be acceptable, too, as long as a second FIN is not emitted (the first FIN may be retransmitted though).

September 1981

Transmission Control Protocol
Functional Specification

CLOSE Call

CLOSE-WAIT STATE

Queue this request until all preceding SENDs have been
segmentized; then send a FIN segment, enter CLOSING state.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Respond with "error: connection closing".

ABORT Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVES should be returned with "error: connection reset" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

All queued SENDs and RECEIVES should be given "connection reset" notification, delete the TCB, enter CLOSED state, and return.

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE

Send a reset segment:

<SEQ=SN.D.NXT><CTL=RST>

All queued SENDs and RECEIVES should be given "connection reset" notification; all segments queued for transmission (except for the RST formed above) or retransmission should be flushed, delete the TCB, enter CLOSED state, and return.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Respond with "ok" and delete the TCB, enter CLOSED state, and return.

September 1981

Transmission Control Protocol
Functional Specification

STATUS Call

STATUS Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Return "state = LISTEN", and the TCB pointer.

SYN-SENT STATE

Return "state = SYN-SENT", and the TCB pointer.

SYN-RECEIVED STATE

Return "state = SYN-RECEIVED", and the TCB pointer.

ESTABLISHED STATE

Return "state = ESTABLISHED", and the TCB pointer.

FIN-WAIT-1 STATE

Return "state = FIN-WAIT-1", and the TCB pointer.

FIN-WAIT-2 STATE

Return "state = FIN-WAIT-2", and the TCB pointer.

CLOSE-WAIT STATE

Return "state = CLOSE-WAIT", and the TCB pointer.

CLOSING STATE

Return "state = CLOSING", and the TCB pointer.

LAST-ACK STATE

Return "state = LAST-ACK", and the TCB pointer.

TIME-WAIT STATE

Return "state = TIME-WAIT", and the TCB pointer.

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

SEGMENT ARRIVES

If the state is CLOSED (i.e., TCB does not exist) then

all data in the incoming segment is discarded. An incoming segment containing a RST is discarded. An incoming segment not containing a RST causes a RST to be sent in response. The acknowledgment and sequence field values are selected to make the reset sequence acceptable to the TCP that sent the offending segment.

If the ACK bit is off, sequence number zero is used,

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the ACK bit is on,

<SEQ=SEG.ACK><CTL=RST>

Return.

If the state is LISTEN then

first check for an RST

An incoming RST should be ignored. Return.

second check for an ACK

Any acknowledgment is bad if it arrives on a connection still in the LISTEN state. An acceptable reset segment should be formed for any arriving ACK-bearing segment. The RST should be formatted as follows:

<SEQ=SEG.ACK><CTL=RST>

Return.

third check for a SYN

If the SYN bit is set, check the security. If the security/compartments on the incoming segment does not exactly match the security/compartments in the TCB then send a reset and return.

<SEQ=SEG.ACK><CTL=RST>

September 1981

SEGMENT ARRIVES

If the SEG.PRC is greater than the TCB.PRC then if allowed by the user and the system set TCB.PRC<-SEG.PRC, if not allowed send a reset and return.

<SEQ=SEG.ACK><CTL=RST>

If the SEG.PRC is less than the TCB.PRC then continue.

Set RCV.NXT to SEG.SEQ+1, IRS is set to SEG.SEQ and any other control or text should be queued for processing later. ISS should be selected and a SYN segment sent of the form:

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

SND.NXT is set to ISS+1 and SND.UNA to ISS. The connection state should be changed to SYN-RECEIVED. Note that any other incoming control or data (combined with SYN) will be processed in the SYN-RECEIVED state, but processing of SYN and ACK should not be repeated. If the listen was not fully specified (i.e., the foreign socket was not fully specified), then the unspecified fields should be filled in now.

fourth other text or control

Any other control or text-bearing segment (not containing SYN) must have an ACK and thus would be discarded by the ACK processing. An incoming RST segment could not be valid, since it could not have been sent in response to anything sent by this incarnation of the connection. So you are unlikely to get here, but if you do, drop the segment, and return.

If the state is SYN-SENT then

first check the ACK bit

If the ACK bit is set

If SEG.ACK =< ISS, or SEG.ACK > SND.NXT, send a reset (unless the RST bit is set, if so drop the segment and return)

<SEQ=SEG.ACK><CTL=RST>

and discard the segment. Return.

If SND.UNA =< SEG.ACK =< SND.NXT then the ACK is acceptable.

second check the RST bit

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

If the RST bit is set

If the ACK was acceptable then signal the user "error: connection reset", drop the segment, enter CLOSED state, delete TCB, and return. Otherwise (no ACK) drop the segment and return.

third check the security and precedence

If the security/compartiment in the segment does not exactly match the security/compartiment in the TCB, send a reset

If there is an ACK

<SEQ=SEG.ACK><CTL=RST>

Otherwise

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If there is an ACK

The precedence in the segment must match the precedence in the TCB, if not, send a reset

<SEQ=SEG.ACK><CTL=RST>

If there is no ACK

If the precedence in the segment is higher than the precedence in the TCB then if allowed by the user and the system raise the precedence in the TCB to that in the segment, if not allowed to raise the prec then send a reset.

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the precedence in the segment is lower than the precedence in the TCB continue.

If a reset was sent, discard the segment and return.

fourth check the SYN bit

This step should be reached only if the ACK is ok, or there is no ACK, and it the segment did not contain a RST.

If the SYN bit is on and the security/compartiment and precedence

SEGMENT ARRIVES

are acceptable then, RCV.NXT is set to SEG.SEQ+1, IRS is set to SEG.SEQ. SND.UNA should be advanced to equal SEG.ACK (if there is an ACK), and any segments on the retransmission queue which are thereby acknowledged should be removed.

If SND.UNA > ISS (our SYN has been ACKed), change the connection state to ESTABLISHED, form an ACK segment

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

and send it. Data or controls which were queued for transmission may be included. If there are other controls or text in the segment then continue processing at the sixth step below where the URG bit is checked, otherwise return.

Otherwise enter SYN-RECEIVED, form a SYN,ACK segment

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

and send it. If there are other controls or text in the segment, queue them for processing after the ESTABLISHED state has been reached, return.

fifth, if neither of the SYN or RST bits is set then drop the segment and return.

September 1981

Transmission Control Protocol
functional Specification

SEGMENT ARRIVES

Otherwise,

first check sequence number

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed.

There are four cases for the acceptability test for an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

If the RCV.WND is zero, no segments will be acceptable, but special allowance should be made to accept valid ACKs, URGs and RSTs.

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the acknowledgment, drop the unacceptable segment and return.

SEGMENT ARRIVES

In the following it is assumed that the segment is the idealized segment that begins at RCV.NXT and does not exceed the window. One could tailor actual segments to fit this assumption by trimming off any portions that lie outside the window (including SYN and FIN), and only processing further if the segment then begins at RCV.NXT. Segments with higher beginning sequence numbers may be held for later processing.

second check the RST bit,

SYN-RECEIVED STATE

If the RST bit is set

If this connection was initiated with a passive OPEN (i.e., came from the LISTEN state), then return this connection to LISTEN state and return. The user need not be informed. If this connection was initiated with an active OPEN (i.e., came from SYN-SENT state) then the connection was refused, signal the user "connection refused". In either case, all segments on the retransmission queue should be removed. And in the active OPEN case, enter the CLOSED state and delete the TCB, and return.

ESTABLISHED
FIN-WAIT-1
FIN-WAIT-2
CLOSE-WAIT

If the RST bit is set then, any outstanding RECEIVES and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT

If the RST bit is set then, enter the CLOSED state, delete the TCB, and return.

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

third check security and precedence

SYN-RECEIVED

If the security/compartments and precedence in the segment do not exactly match the security/compartments and precedence in the TCB then send a reset, and return.

ESTABLISHED STATE

If the security/compartments and precedence in the segment do not exactly match the security/compartments and precedence in the TCB then send a reset, any outstanding RECEIVES and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

Note this check is placed following the sequence check to prevent a segment from an old connection between these ports with a different security or precedence from causing an abort of the current connection.

fourth, check the SYN bit,

SYN-RECEIVED
ESTABLISHED STATE
FIN-WAIT STATE-1
FIN-WAIT STATE-2
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

If the SYN is in the window it is an error, send a reset, any outstanding RECEIVES and SEND should receive "reset" responses, all segment queues should be flushed, the user should also receive an unsolicited general "connection reset" signal, enter the CLOSED state, delete the TCB, and return.

If the SYN is not in the window this step would not be reached and an ack would have been sent in the first step (sequence number check).

fifth check the ACK field,

if the ACK bit is off drop the segment and return

if the ACK bit is on

SYN-RECEIVED STATE

If $SND.UNA \leq SEG.ACK \leq SND.NXT$ then enter ESTABLISHED state and continue processing.

If the segment acknowledgment is not acceptable, form a reset segment,

$\langle SEQ=SEG.ACK \rangle \langle CTL=RST \rangle$

and send it.

ESTABLISHED STATE

If $SND.UNA < SEG.ACK \leq SND.NXT$ then, set $SND.UNA \leftarrow SEG.ACK$. Any segments on the retransmission queue which are thereby entirely acknowledged are removed. Users should receive positive acknowledgments for buffers which have been SENT and fully acknowledged (i.e., SEND buffer should be returned with "ok" response). If the ACK is a duplicate ($SEG.ACK < SND.UNA$), it can be ignored. If the ACK acks something not yet sent ($SEG.ACK > SND.NXT$) then send an ACK, drop the segment, and return.

If $SND.UNA < SEG.ACK \leq SND.NXT$, the send window should be updated. If ($SND.WL1 < SEG.SEQ$ or ($SND.WL1 = SEG.SEQ$ and $SND.WL2 \leq SEG.ACK$)), set $SND.WND \leftarrow SEG.WND$, set $SND.WL1 \leftarrow SEG.SEQ$, and set $SND.WL2 \leftarrow SEG.ACK$.

Note that $SND.WND$ is an offset from $SND.UNA$, that $SND.WL1$ records the sequence number of the last segment used to update $SND.WND$, and that $SND.WL2$ records the acknowledgment number of the last segment used to update $SND.WND$. The check here prevents using old segments to update the window.

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

FIN-WAIT-1 STATE

In addition to the processing for the ESTABLISHED state, if our FIN is now acknowledged then enter FIN-WAIT-2 and continue processing in that state.

FIN-WAIT-2 STATE

In addition to the processing for the ESTABLISHED state, if the retransmission queue is empty, the user's CLOSE can be acknowledged ("ok") but do not delete the TCB.

CLOSE-WAIT STATE

Do the same processing as for the ESTABLISHED state.

CLOSING STATE

In addition to the processing for the ESTABLISHED state, if the ACK acknowledges our FIN then enter the TIME-WAIT state, otherwise ignore the segment.

LAST-ACK STATE

The only thing that can arrive in this state is an acknowledgment of our FIN. If our FIN is now acknowledged, delete the TCB, enter the CLOSED state, and return.

TIME-WAIT STATE

The only thing that can arrive in this state is a retransmission of the remote FIN. Acknowledge it, and restart the 2 MSL timeout.

sixth, check the URG bit,

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

If the URG bit is set, $RCV.UP \leftarrow \max(RCV.UP, SEG.UP)$, and signal the user that the remote side has urgent data if the urgent pointer (RCV.UP) is in advance of the data consumed. If the user has already been signaled (or is still in the "urgent mode") for this continuous sequence of urgent data, do not signal the user again.

SEGMENT ARRIVES

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT

This should not occur, since a FIN has been received from the remote side. Ignore the URG.

seventh, process the segment text,

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

Once in the ESTABLISHED state, it is possible to deliver segment text to user RECEIVE buffers. Text from segments can be moved into buffers until either the buffer is full or the segment is empty. If the segment empties and carries an PUSH flag, then the user is informed, when the buffer is returned, that a PUSH has been received.

When the TCP takes responsibility for delivering the data to the user it must also acknowledge the receipt of the data.

Once the TCP takes responsibility for the data it advances RCV.NXT over the data accepted, and adjusts RCV.WND as appropriate to the current buffer availability. The total of RCV.NXT and RCV.WND should not be reduced.

Please note the window management suggestions in section 3.7.

Send an acknowledgment of the form:

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

This acknowledgment should be piggybacked on a segment being transmitted if possible without incurring undue delay.

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

This should not occur, since a FIN has been received from the remote side. Ignore the segment text.

eighth, check the FIN bit.

Do not process the FIN if the state is CLOSED, LISTEN or SYN-SENT since the SEG.SEQ cannot be validated; drop the segment and return.

If the FIN bit is set, signal the user "connection closing" and return any pending RECEIVES with same message, advance RCV.NXT over the FIN, and send an acknowledgment for the FIN. Note that FIN implies PUSH for any segment text not yet delivered to the user.

SYN-RECEIVED STATE
ESTABLISHED STATE

Enter the CLOSE-WAIT state.

FIN-WAIT-1 STATE

If our FIN has been ACKed (perhaps in this segment), then enter TIME-WAIT, start the time-wait timer, turn off the other timers; otherwise enter the CLOSING state.

FIN-WAIT-2 STATE

Enter the TIME-WAIT state. Start the time-wait timer, turn off the other timers.

CLOSE-WAIT STATE

Remain in the CLOSE-WAIT state.

CLOSING STATE

Remain in the CLOSING state.

LAST-ACK STATE

Remain in the LAST-ACK state.

TIME-WAIT STATE

Remain in the TIME-WAIT state. Restart the 2 MSL time-wait
timeout.

and return.

September 1981

Transmission Control Protocol
Functional Specification

USER TIMEOUT

USER TIMEOUT

For any state if the user timeout expires, flush all queues, signal the user "error: connection aborted due to user timeout" in general and for any outstanding calls, delete the TCB, enter the CLOSED state and return.

RETRANSMISSION TIMEOUT

For any state if the retransmission timeout expires on a segment in the retransmission queue, send the segment at the front of the retransmission queue again, reinitialize the retransmission timer, and return.

TIME-WAIT TIMEOUT

If the time-wait timeout expires on a connection delete the TCB, enter the CLOSED state and return.

GLOSSARY

1822

BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ACK

A control bit (acknowledge) occupying no sequence space, which indicates that the acknowledgment field of this segment specifies the next sequence number the sender of this segment is expecting to receive, hence acknowledging receipt of all previous sequence numbers.

ARPANET message

The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet

A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

connection

A logical communication path identified by a pair of sockets.

datagram

A message sent in a packet switched computer communications network.

Destination Address

The destination address, usually the network and host identifiers.

FIN

A control bit (finis) occupying one sequence number, which indicates that the sender will send no more data or control occupying sequence space.

fragment

A portion of a logical unit of data, in particular an internet fragment is a portion of an internet datagram.

FTP

A file transfer protocol.

September 1981

Transmission Control Protocol
Glossary

header

Control information at the beginning of a message, segment, fragment, packet or block of data.

host

A computer. In particular a source or destination of messages from the point of view of the communication network.

Identification

An Internet Protocol field. This identifying value assigned by the sender aids in assembling the fragments of a datagram.

IMP

The Interface Message Processor, the packet switch of the ARPANET.

internet address

A source or destination address specific to the host level.

internet datagram

The unit of data exchanged between an internet module and the higher level protocol together with the internet header.

internet fragment

A portion of the data of an internet datagram with an internet header.

IP

Internet Protocol.

IRS

The Initial Receive Sequence number. The first sequence number used by the sender on a connection.

ISN

The Initial Sequence Number. The first sequence number used on a connection, (either ISS or IRS). Selected on a clock based procedure.

ISS

The Initial Send Sequence number. The first sequence number used by the sender on a connection.

leader

Control information at the beginning of a message or block of data. In particular, in the ARPANET, the control information on an ARPANET message at the host-IMP interface.

left sequence

This is the next sequence number to be acknowledged by the data receiving TCP (or the lowest currently unacknowledged sequence number) and is sometimes referred to as the left edge of the send window.

local packet

The unit of transmission within a local network.

module

An implementation, usually in software, of a protocol or other procedure.

MSL

Maximum Segment Lifetime, the time a TCP segment can exist in the internetwork system. Arbitrarily defined to be 2 minutes.

octet

An eight bit byte.

Options

An Option field may contain several options, and each option may be several octets in length. The options are used primarily in testing situations; for example, to carry timestamps. Both the Internet Protocol and TCP provide for options fields.

packet

A package of data with a header which may or may not be logically complete. More often a physical packaging than a logical packaging of data.

port

The portion of a socket that specifies which logical input or output channel of a process is associated with the data.

process

A program in execution. A source or destination of data from the point of view of the TCP or other host-to-host protocol.

PUSH

A control bit occupying no sequence space, indicating that this segment contains data that must be pushed through to the receiving user.

RCV.NXT

receive next sequence number

Transmission Control Protocol
Glossary

RCV.UP

receive urgent pointer

RCV.WND

receive window

receive next sequence number

This is the next sequence number the local TCP is expecting to receive.

receive window

This represents the sequence numbers the local (receiving) TCP is willing to receive. Thus, the local TCP considers that segments overlapping the range RCV.NXT to RCV.NXT + RCV.WND - 1 carry acceptable data or control. Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded.

RST

A control bit (reset), occupying no sequence space, indicating that the receiver should delete the connection without further interaction. The receiver can determine, based on the sequence number and acknowledgment fields of the incoming segment, whether it should honor the reset command or ignore it. In no case does receipt of a segment containing RST give rise to a RST in response.

RTP

Real Time Protocol: A host-to-host protocol for communication of time critical information.

SEG.ACK

segment acknowledgment

SEG.LEN

segment length

SEG.PRC

segment precedence value

SEG.SEQ

segment sequence

SEG.UP

segment urgent pointer field

SEG.WND

segment window field

segment

A logical unit of data, in particular a TCP segment is the unit of data transferred between a pair of TCP modules.

segment acknowledgment

The sequence number in the acknowledgment field of the arriving segment.

segment length

The amount of sequence number space occupied by a segment, including any controls which occupy sequence space.

segment sequence

The number in the sequence field of the arriving segment.

send sequence

This is the next sequence number the local (sending) TCP will use on the connection. It is initially selected from an initial sequence number curve (ISN) and is incremented for each octet of data or sequenced control transmitted.

send window

This represents the sequence numbers which the remote (receiving) TCP is willing to receive. It is the value of the window field specified in segments from the remote (data receiving) TCP. The range of new sequence numbers which may be emitted by a TCP lies between SND.NXT and $\text{SND.UNA} + \text{SND.WND} - 1$. (Retransmissions of sequence numbers between SND.UNA and SND.NXT are expected, of course.)

SND.NXT

send sequence

SND.UNA

left sequence

SND.UP

send urgent pointer

SND.WL1

segment sequence number at last window update

SND.WL2

segment acknowledgment number at last window update

Transmission Control Protocol
Glossary

SND.WND

send window

socket

An address which specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port.

Source Address

The source address, usually the network and host identifiers.

SYN

A control bit in the incoming segment, occupying one sequence number, used at the initiation of a connection, to indicate where the sequence numbering will start.

TCB

Transmission control block, the data structure that records the state of a connection.

TCB.PRC

The precedence of the connection.

TCP

Transmission Control Protocol: A host-to-host protocol for reliable communication in internetwork environments.

TOS

Type of Service, an Internet Protocol field.

Type of Service

An Internet Protocol field which indicates the type of service for this internet fragment.

URG

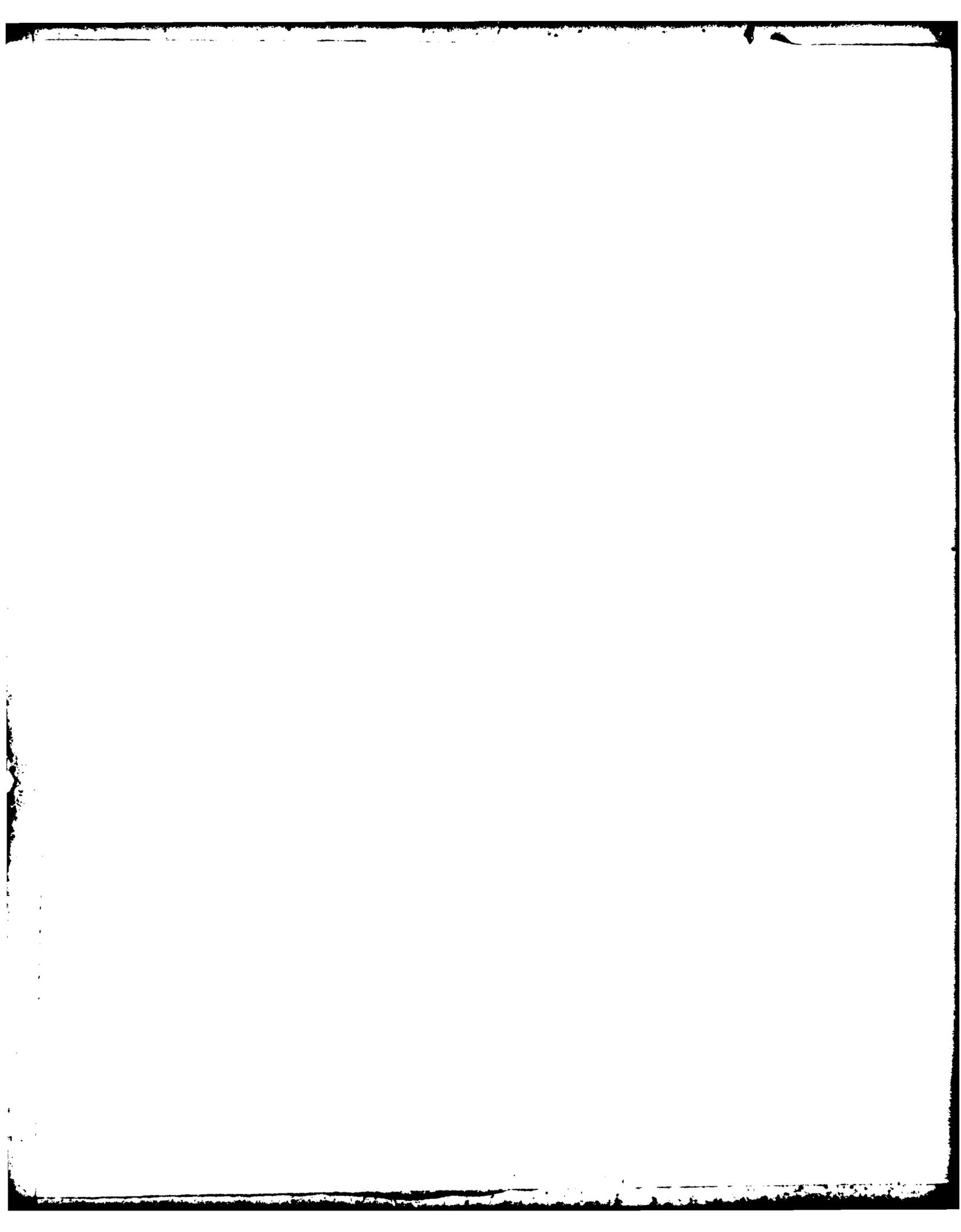
A control bit (urgent), occupying no sequence space, used to indicate that the receiving user should be notified to do urgent processing as long as there is data to be consumed with sequence numbers less than the value indicated in the urgent pointer.

urgent pointer

A control field meaningful only when the URG bit is on. This field communicates the value of the urgent pointer which indicates the data octet associated with the sending user's urgent call.

REFERENCES

- [1] Cerf, V., and R. Kahn, "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, Vol. COM-22, No. 5, pp 637-648, May 1974.
- [2] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981.
- [3] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols", Computer Networks, Vol. 2, No. 6, pp. 454-473, December 1978.
- [4] Postel, J., "Assigned Numbers", RFC 790, USC/Information Sciences Institute, September 1981.



PRE-EMPTION

In circuit-switching systems, once a user has acquired a circuit, the communication bandwidth of that circuit is dedicated, even if it is not used. When the system saturates, additional circuit set-up requests are blocked. To allow high precedence users to gain access to circuit resources, systems such as AUTOVON associate a precedence with each telephone instrument. Those instruments with high precedence can pre-empt circuit resources, causing lower precedence users to be cut off.

In message switching systems such as AUTODIN I, incoming traffic is stored on disks (or drums or tape) and processed in order of precedence. If a high precedence message is entered into the system, it is processed and forwarded as quickly as possible. When the high precedence message arrives at the destination message switch, it may pre-empt the use of the output devices on the switch, interrupting the printing of a lower precedence message.

In packet switching systems, there is little or no storage in the transport system so that precedence has little impact on delay for processing a packet. However, when a packet switching system reaches saturation, it rejects offered traffic. Precedence can be used in saturated packet switched systems to sort traffic queued for entry into the system.

In general, precedence is a tool for deciding how to allocate resources when systems are saturated. In circuit switched systems, the resource is circuits; in message switched systems the resource is the message switch processor; and in packet switching the resource is the packet switching system itself.

This capability can be realized in AUTODIN II without adding any new mechanisms to TCP (except to make precedence of incoming connection requests visible to the processes which use TCP). To allow pre-emptive access to a particular terminal, the software (i.e., THP) which supports terminal access to the TAC can be configured so as to always have a LISTEN posted for that terminal, even if the terminal has a connection in operation. For example in the ARPANET TENEX systems, the user TELNET permits a user to have many connections open at one time - the user can switch among them at will. To the extent that this can be done without violating security requirements, one could imagine a multi-connection THP which always leaves a LISTEN pending for incoming connection requests. If a connection is established, the THP can decide, based on its precedence, whether to pre-empt any existing connection and to switch the user to the high precedence one.

If the user is working with several connections of different precedence at the same time, the THP would close or abort the lowest precedence

September 1981

Pre-Emption

connection in favor of the higher precedence pre-empting one. Then the IHP would do a new LISTEN on that terminal's port in case a higher precedence connection is attempted.

One of the reasons for suggesting this model is that processes are the users of TCP (in general) and that TCP itself cannot cause processes to be created on behalf of an incoming connection request. Implementations could be realized in which TCPs accept incoming connection requests and, based on the destination port number, create appropriate server processes. In terms of pre-empting access to a remote terminal, however, it seems more sensible to let the process which interfaces the terminal to the system mediate the pre-emption. If the terminal is not connected or is turned off, there is no point in creating a process to serve the incoming high precedence connection request.

For example, suppose a routine FTP is in operation between Host X and Host Y. Host Z decides to do a flash-override FTP to Host X. It opens a high precedence connection via its TCP and the "SYN" goes out to the FTP port on Host X.

FTP always leaves one LISTEN pending to pre-empt lower precedence remote users if it cannot serve one more user (and still keep a LISTEN pending). In this way, the FTP is naturally in a state permitting the high precedence connection request to be properly served, and the FTP can initiate any cleaning up that is needed to deal with the pre-emption.

In general, this strategy permits the processes using TCP to accommodate pre-emption in the context of the applications they support.

A non-pre-emptable process is one that does not have a LISTEN pending while it is serving one (or more) users.

The actions taken to deal with pre-emption of TCP connections will be application-process specific and this strategy of a second (or N+1st) LISTEN is well suited to the situation.

Pre-emption may also be necessary at the site initiating a high precedence connection request. Suppose there is a high precedence user who wants to open an FTP connection request from Host Z to Host X. But all FTP and/or TCP resources are saturated when this user tries to start the user FTP process. In this case, the operating system would have to know about the precedence of the user and would have to locally pre-empt resources on his behalf (e.g., by logging out lower precedence users). This is a system issue, not specific only to TCP. Implementation of pre-emption at the source could vary greatly. Precedence may be associated with a user or with a terminal. The TCP implementation may locally pre-empt resources to serve high precedence users. The operating system may make all pre-emption decisions.

SERVICE MAPPINGS

This memo describes the relationship between the Internet Protocol (IP) [1] Type of Service and the service parameters of specific networks.

The IP Type of Service has the following fields:

Bits 0-2: Precedence.
Bit 3: 0 = Normal Delay, 1 = Low Delay.
Bits 4: 0 = Normal Throughput, 1 = High Throughput.
Bits 5: 0 = Normal Reliability, 1 = High Reliability.
Bit 6-7: Reserved for Future Use.

0	1	2	3	4	5	6	7
PRECEDENCE			D	T	R	0	0

111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine

The individual networks listed here have very different and specific service choices.

AUTODIN II

The service choices are in two parts: Traffic Acceptance Categories, and Application Type. The Traffic Acceptance Categories can be mapped into and out of the IP TOS precedence reasonably directly. The Application types can be mapped into the remaining IP TOS fields as follows.

TA	DELAY	THROUGHPUT	RELIABILITY
---	---	-----	-----
I/A	1	0	0
Q/R	0	0	0
B1	0	1	0
B2	0	1	1
DTR	TA		
---	---		
000	Q/R		
001	Q/R		
010	B1		
011	B2		
100	I/A		
101	I/A		
110	I/A		
111	error		

ARPANET

The service choices are in quite limited. There is one priority bit that can be mapped to the high order bit of the IP TOS precedence. The other choices are to use the regular ("Type 0") messages vs. the uncontrolled ("Type 3") messages, or to use single packet vs. multipacket messages. The mapping of ARPANET parameters into IP TOS parameters can be as follows.

Type	Size	DELAY	THROUGHPUT	RELIABILITY
----	----	----	----	----
0	S	1	0	0
0	M	0	0	0
3	S	1	0	0
3	M	not allowed		

DTR	Type	Size
----	----	----
000	0	M
001	0	M
010	0	M
011	0	M
100	3	S
101	0	S
110	3	S
111	error	

PRNET

There is no priority indication. The two choices are to use the station routing vs. point-to-point routing, or to require acknowledgments vs. having no acknowledgments. The mapping of PRNET parameters into IP TOS parameters can be as follows.

Routing	Acks	DELAY	THROUGHPUT	RELIABILITY
-----	-----	-----	-----	-----
ptp	no	1	0	0
ptp	yes	1	0	1
station	no	0	0	0
station	yes	0	0	1

DTR	Routing	Acks
---	-----	-----
000	station	no
001	station	yes
010	station	no
011	station	yes
100	ptp	no
101	ptp	yes
110	ptp	no
111	ptp	yes

SATNET

There is no priority indication. The four choices are to use the block vs. stream type, to select one of four delay categories, to select one of two holding time strategies, or to request one of three reliability levels. The mapping of SATNET parameters into IP TOS parameters can thus quite complex there being $2*4*2*3=48$ distinct possibilities.

References

- [1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

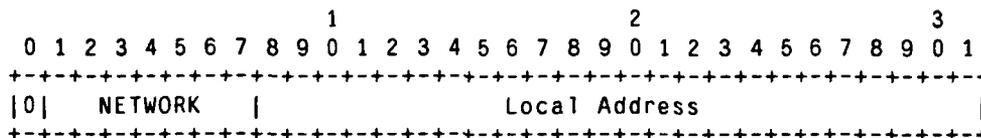
ADDRESS MAPPINGS

Internet Addresses

This memo describes the relationship between address fields used in the Internet Protocol (IP) [1] and several specific networks.

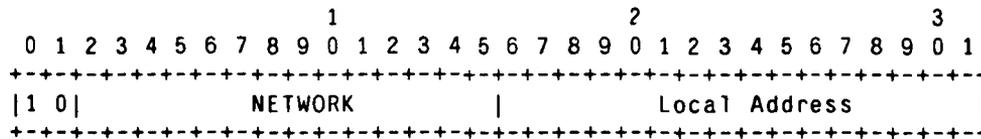
An internet address is a 32 bit quantity, with several codings as shown below.

The first type (or class a) of address has a 7-bit network number and a 24-bit local address.



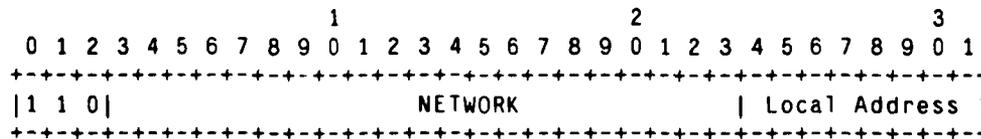
Class A Address

The second type (or class b) of address has a 14-bit network number and a 16-bit local address.



Class B Address

The third type (or class c) of address has a 21-bit network number and a 8-bit local address.



Class C Address

The local address carries information to address a host in the network identified by the network number. Since each network has a

particular address format and length, the following section describes the mapping between internet local addresses and the actual address format used in the particular network.

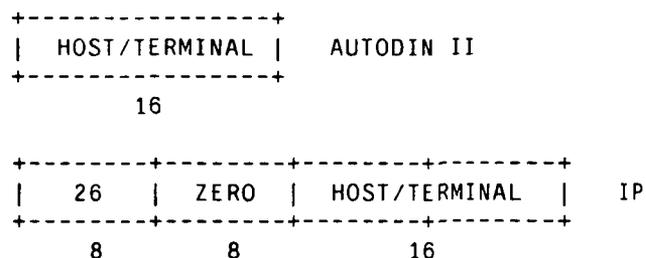
Internet to Local Net Address Mappings

The following transformations are used to convert internet addresses to local net addresses and vice versa:

AUTODIN II

The AUTODIN II has 16 bit subscriber addresses which identify either a host or a terminal. These addresses may be assigned independent of location. The 16 bit AUTODIN II address is located in the 24 bit internet local address as shown below.

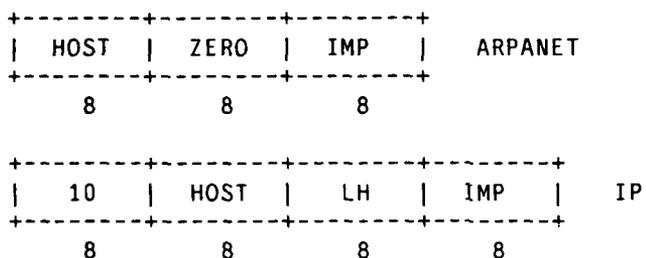
The network number of the AUTODIN II is 26 (Class A).



ARPANET

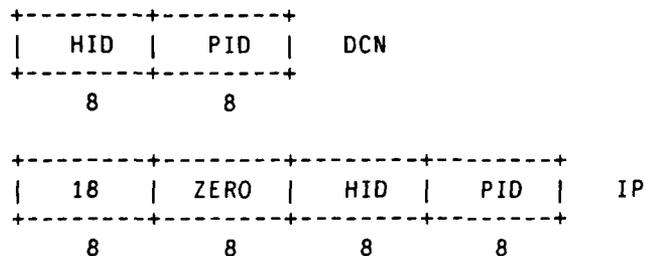
The ARPANET (with 96 bit leaders) has 24 bit addresses. The 24 bits are assigned to host, logical host, and IMP leader fields as illustrated below. These 24 bit addresses are used directly for the 24 bit local address of the internet address. However, the ARPANET IMPs do not yet support this form of logical addressing so the logical host field is set to zero in the leader.

The network number of the ARPANET is 10 (Class A).

DCNs

The Distributed Computing Networks (DCNs) at COMSAT and UCL use 16 bit addresses divided into an 8 bit host identifier (HID), and an 8 bit process identifier (PID). The format locates these 16 bits in the low order 16 bits of the 24 bit internet address, as shown below.

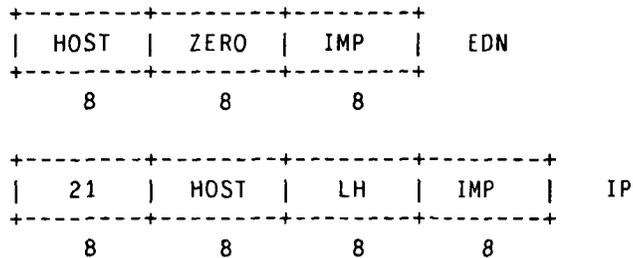
The network number of the COMSAT-DCN is 29 (Class A), and of the UCL-DCN is 30 (Class A).



EDN

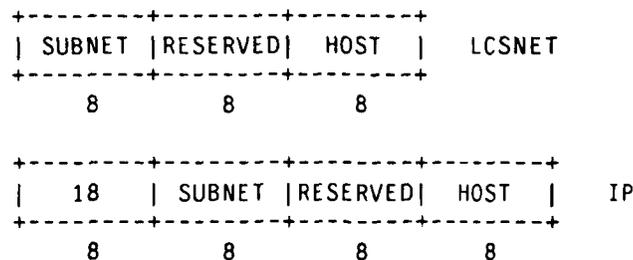
The Experimental Data Network at the Defense Communication Engineering Center (DCEC) uses the same type of addresses as the ARPANET (with 96 bit leaders) and has 24 bit addresses. The 24 bits are assigned to host, logical host, and IMP leader fields as illustrated below. These 24 bit addresses are used directly for the 24 bit local address of the internet address. However, the IMPs do not yet support this form of logical addressing so the logical host field is set to zero in the leader.

The network number of the EDN is 21 (Class A).

LCSNET

The LCS NET at MIT's Laboratory for Computer Science uses 32 bit addresses of several formats. Please see [3] for more details. The most common format locates the low order 24 bits of the 32 bit LCS NET address in the 24 bit internet local address, as shown below.

The network number of the LCS NET is 18 (Class A).

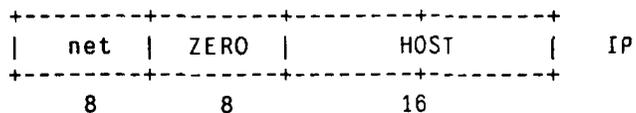
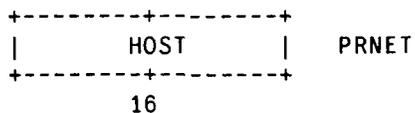


PRNET

The Packet Radio networks use 16 bit addresses. These are independent of location (indeed the hosts may be mobile). The 16 bit PRNET addresses are located in the 24 bit internet local address as shown below.

The network numbers of the PRNETs are:

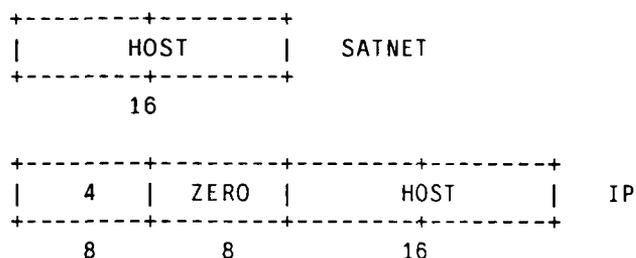
BBN-PR	1 (Class A)
SF-PR-1	2 (Class A)
SILL-PR	5 (Class A)
SF-PR-2	6 (Class A)
BRAGG-PR	9 (Class A)
DC-PR	20 (Class A)



SATNET

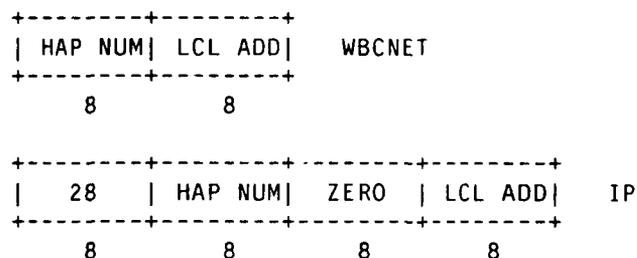
The Atlantic Satellite Packet Network has 16 bit addresses for hosts. These addresses may be assigned independent of location (i.e., ground station). It is also possible to assign several addresses to one physical host, so the addresses are logical addresses. The 16 bit SATNET address is located in the 24 bit internet local address as shown below.

The network number of the SATNET is 4 (Class A).

WBCNET

The Wideband Communication Satellite Packet Network (WBCNET) Host Access Protocol (HAP) has 16 bit addresses for hosts. It is possible to assign several addresses to one physical host, so the addresses are logical addresses. The 16 bit WBCNET address is divided into a HAP Number field and a Local Address field, and is located in the 24 bit internet local address as shown below. Please see [2] for more details.

The network number of the WBCNET is 28 (Class A).



References

- [1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.
- [2] Pershing J., "Addressing Revisited," Bolt Beranek and Newman Inc., W Note 27, May 1981.
- [3] Noel Chiappa, David Clark, David Reed, "LCS Net Address Format," M.I.T. Laboratory for Computer Science Network Implementation, Note No.5, IEN 82, February 1979.

AWD